Package: stepPenal (via r-universe)

October 17, 2024

Type Package

Title Stepwise Forward Variable Selection in Penalized Regression

Version 0.2

Author Eleni Vradi

Maintainer Eleni Vradi <vradi.eleni@gmail.com>

Description Model Selection Based on Combined Penalties. This package implements a stepwise forward variable selection algorithm based on a penalized likelihood criterion that combines the L0 with L2 or L1 norms.

License GPL-2

LazyData TRUE

Depends R (>= 3.5.0)

Imports glmnet, mvtnorm, pROC, dfoptim, caret, stats, base

NeedsCompilation no

Encoding UTF-8

RoxygenNote 6.1.0

Date/Publication 2018-08-24 20:50:03 UTC

Repository https://veleni.r-universe.dev

RemoteUrl https://github.com/cran/stepPenal

RemoteRef HEAD

RemoteSha 7cc542f036f64d86a5cff06b7d48286c47301bf7

Contents

`findROC` *Compute the area under the ROC curve*

Description

This function computes the numeric value of area under the ROC curve (AUC) with the trapezoidal rule. It is a wrapper function around the pRoc function in the roc package

Usage

```
findROC(Data, coeff)
```

Arguments

`Data`	a data matrix; in the first column there should be the binary response variable y. If you give the training dataset it will calculate the in-sample AUC. If supplied with a new dataset then it will return the predictive AUC.
`coeff`	vector of coefficients

Value

The area under the ROC curve, the sensitivity and specificity

See Also

`roc`

Examples

```
## Not run:
set.seed(14)
beta    <- c(3, 2, -1.6, -4)
noise   <- 5
simData <- SimData(N=100,beta=beta, noise=noise, corr=FALSE)

stepPenal<- StepPenal(Data=simData, lamda=1.2, w=0.7)

(coeffP <- stepPenal$coeffP)

findROC(simData, coeff=coeffP)

## End(Not run)
```

`lassomodel` *Fits a lasso model and a lasso followed by a stepAIC algorithm.*

Description

Fits a lasso model and a lasso followed by a stepAIC algorithm.

Usage

```
lassomodel(Data, standardize = TRUE, measure = c("deviance"),
  nfold = 5)
```

Arguments

`Data`	a data frame, as a first column should have the response variable y
`standardize`	Logical flag for variable standardization, prior to fitting the model. Default is standardize=TRUE. If variables are in the same units already, you might not wish to standardize.
`measure`	loss to use for cross-validation. measure="auc" is for two-class logistic regression only, and gives area under the ROC curve. measure="deviance", uses the deviance for logistic regression.
`nfold`	number of folds - default is 5. Although nfolds can be as large as the sample size (leave-one-out CV), it is not recommended for large datasets. Smallest value allowable is nfolds=3

Details

the function lassomodel is a wrapper function over the glmnet::glmnet. The parameter lambda is tuned by 10-fold cross-validation with the glmnet::cv.glmnet function. The selected lambda is the one that gives either the minimum deviance (measure="deviance") or the maximum auc (measure="auc") or minimum misclassification error (measure="class")

Value

a list with the coefficients in the final model for the lasso fit and also for the lasso followed by stepAIC.

See Also

`glmnet`

Examples

```
## Not run:
set.seed(14)
beta    <- c(3, 2, -1.6, -4)
noise   <- 5
simData <- SimData(N=100, beta=beta, noise=noise, corr=FALSE)

lassofit <- lassomodel(Data=simData, measure="auc")
lassofit

lassofit2 <- lassomodel(Data=simData, measure="deviance")
lassofit2

## End(Not run)
```

`objFun` *Objective function*

Description

Objective (non-convex) function to minimize (objFun=-logL+ lamda*CL, CL= (1-w)L0 + wL1)

Usage

```
objFun(x, y, lamda, w, beta, epsilon)
```

Arguments

`x`	input matrix, of dimension nobs x nvars. Each row is an observation vector.
`y`	binary response variable
`lamda`	a tuning penalty parameter
`w`	the weighting parameter for L1; then (1-w) is the weight for L0
`beta`	coefficients
`epsilon`	the continuity parameter

Value

the value of the objective function evaluated at the given points.

Examples

```
set.seed(14)
beta    <- c(3, 2, -1.6, -1)
noise   <- 5
simData <- SimData(N=100, beta=beta, noise=noise, corr=FALSE)
```

```
x  <- as.matrix(simData[,-1][,1])
y  <- as.matrix(simData$y)
betapoints <- seq(-2,2,0.01)

lamda <- 1
w     <- 0.6
epsilon <- 0.1

out <- numeric(length(betapoints))
for(i in 1:length(betapoints)){
 out[i]<- objFun(x, y, lamda=lamda, w=w, beta=betapoints[i], epsilon=epsilon)
}
plot(betapoints, out, type="l", ylab="objFun")
```

`optimPenaLik` *Variable selection based on the combined penalty CL= (1-w)L0 + wL1*

Description

Methods to use for optimization include Hooke-Jeeves derivative-free minimization algorithm (hjk), or the BFGS method (modified Quasi-Newton). This method does variable selection by shrinking the coefficients towards zero using the combined penalty (CL= (1-w)L0 + wL1).

Usage

```
optimPenaLik(Data, lamda, w, standardize = TRUE, algorithms = c("QN",
  "hjk"))
```

Arguments

`Data`	should have the following structure: the first column must be the response variable y
`lamda`	tuning penalty parameter
`w`	the weight parameter for the sum (1-w)L0+ wL1
`standardize`	standardize Logical flag for x variable standardization, prior to fitting the model sequence. The coefficients are always returned on the original scale. Default is standardize=TRUE
`algorithms`	select between BFGS ('QN') or Hooke-Jeeves (hjk) algorithm.

Details

it is recommended to use the tuneParam function to tune parameters lamda and w prior using the optimPenaLik function.

Value

a list with the shrinked coefficients and the names of the selected variables, i.e those variables with estimated coefficient different from zero.

See Also

optim

Examples

```
# use the optimPenaLik function on a simulated dataset, with given lamda and w.
## Not run:
set.seed(14)
beta    <- c(3, 2, -1.6, -1)
noise   <- 5
simData <- SimData(N=100, beta=beta, noise=noise, corr=TRUE)

# use BFGS

before  <- Sys.time()
PenalQN <- optimPenaLik(Data=simData, lamda=1.5, w=0.7,
                    algorithms=c("QN"))
(tot <- Sys.time()-before)
PenalQN


# use Hooke-Jeeves algorithm

before  <- Sys.time()
Penalhjk <- optimPenaLik(Data=simData, lamda=1.5, w=0.7,
                      algorithms=c("hjk"))
(totRun  <- Sys.time() - before)
# total run of approx 0.25sec

Penalhjk

## End(Not run)
```

`optimPenaLikL2` *Variable selection based on the combined penalty CL2= (1-w)L0 + wL2*

Description

Methods to use for optimization include the Hooke-Jeeves derivative-free minimization algorithm (hjk), and the BFGS method (modified Quasi-Newton). This algorithm does variable selection by shrinking the coefficients towards zero using the combined penalty (CL2= (1-w)L0 + wL2).

Usage

```
optimPenaLikL2(Data, lamda, w, standardize = TRUE, algorithms = c("QN",
  "hjk"))
```

Arguments

Data	should have the following structure: the first column must be the response variable y
lamda	tuning penalty parameter
w	the weight parameter for the sum (1-w)L0+ wL2
standardize	standardize Logical flag for x variable standardization, prior to fitting the model sequence. The coefficients are always returned on the original scale. Default is standardize=TRUE
algorithms	select between Simulated annealing or Differential evolution

Details

it is recommended to use the tuneParam function to tune parameters lamda and w prior using the optimPenaLik function.

Value

a list with the shrinked coefficients and the names of the selected variables, i.e those variables with estimated coefficient different from zero.

Examples

```
## Not run:
# use the optimPenaLik function on a simulated dataset, with given lamda and w.
set.seed(14)
beta    <- c(3, 2, -1.6, -1)
noise   <- 5
simData <- SimData(N=100, beta=beta, noise=noise, corr=TRUE)

# example with Quasi-Newton:
before <- Sys.time()
PenalQN <- optimPenaLikL2(Data=simData, lamda=2, w=0.6,
                     algorithms=c("QN"))
after <- Sys.time()
after-before
PenalQN

## End(Not run)
```

---

penalBrier — *Evaluation of the performance of risk prediction models with binary status response variable.*

Description

Evaluation of the performance of risk prediction models with binary status response variable.

Usage

```
penalBrier(Data, coeffP)
```

Arguments

Data	a data matrix; in the first column there should be the response variable y. If you give the training dataset it will calculate the Brier score.
coeffP	a named vector of coefficients

Details

Brier score is a measure for classification performance of a binary classifier. Its values range between [0,1] and the closest is to 0 the better the classifier is. The area under the curve and the Brier score is used to summarize and compare the performance.

Value

the Brier score (misclassification error)

References

Brier, G. W. (1950). Verification of forecasts expressed in terms of probability. Monthly Weather Review 78.

Examples

```
# use the penalBrier function on a simulated dataset, with given lamda and w.
## Not run:
set.seed(14)
beta    <- c(3, 2, -1.6, -4)
noise   <- 5
simData <- SimData(N=100,beta=beta, noise=noise, corr=FALSE)

before   <- Sys.time()
stepPenal<- StepPenal(Data=simData, lamda=1.2, w=0.4)
(totRun  <- Sys.time() - before)

(coeff<- stepPenal$coeffP)
 me <- penalBrier(simData,coeff)

## End(Not run)
```

SimData *Simulate data with normally distributed predictors and binary response*

Description

Simulate data with normally distributed predictors and binary response

Usage

```
SimData(N, beta, noise, corr = TRUE, corr.effect = 0.5)
```

Arguments

N	sample size
beta	coefficients (effect of informative predictors)
noise	variables (effect of uninformative predictors)
corr	Logical, if FALSE the function generates uncorrelated predictors, if TRUE the correlation between predictors is 0.5 by default and the user can supply a different value in the corr.effect argument.
corr.effect	the correlation between informative predictors.

Details

The response y follows a Binomial distribution with probability= exp(X*beta)/(1+exp(X*beta))

Value

A data frame N x p, where p is the total number of informative and uninformative predictors. The first column of the dataframe is the binary response variable y

Examples

```
# simulate data with N=100 (sample size) and 23 predictors; 4 informative and 20 noise

set.seed(14)
beta    <- c(3, 2, -1.6, -4)
noise <- 5
N     <- 100
simData <- SimData(N=N, beta=beta, noise=noise, corr=FALSE)
```

`stepaic` *Stepwise forward variable selection based on the AIC criterion*

Description

It is a wrapper function over the step function in the buildin package stats

Usage

```
stepaic(Data, standardize = TRUE)
```

Arguments

`Data`	a data frame, as a first column should hava the response variable y
`standardize`	Logical flag for x variable standardization, prior to fitting the model sequence. Default is standardize=TRUE

Value

a list with the coefficients of the final model. It also returns the in-sample AUC and the Brier score

See Also

`step`

Examples

```
## Not run:
set.seed(14)
beta    <- c(3, 2, -1.6, -4)
noise   <- 5
simData <- SimData(N=100, beta=beta, noise=noise, corr=FALSE)

stepaicfit <- stepaic(Data=simData)
stepaicfit

## End(Not run)
```

`StepPenal` *Stepwise forward variable selection using penalized regression.*

Description

Stepwise forward variable selection based on the combination of L1 and L0 penalties. The optimization is done using the "BFGS" method in stats::optim

Usage

```
StepPenal(Data, lamda, w, standardize = TRUE)
```

Arguments

`Data`	should have the following structure: the first column must be the binary response variable y.
`lamda`	the tuning penalty parameter
`w`	the weight parameter for the sum (1-w)L0+ wL1
`standardize`	Logical flag for the predictors' standardization, prior to fitting the model. Default is standardize=TRUE

Details

lamda and w parameters need to be tuned by cross-Validation using stepPenal::tuneParam

Value

a list with the shrinked coefficients and the names of the selected variables, i.e those variables with an estimated coefficient different from zero. It also returns the value of the objective function, evaluated for the values of the coefficients.

References

Vradi E, Brannath W, Jaki T, Vonk R. Model selection based on combined penalties for biomarker identification. Journal of biopharmaceutical statistics. 2018 Jul 4;28(4):735-49.

See Also

`optim`

Examples

```
# use the StepPenal function on a simulated dataset, with given lamda and w.

set.seed(14)
beta    <- c(3, 2, -1.6, -1)
noise   <- 5
simData <- SimData(N=100, beta=beta, noise=noise, corr=FALSE)
```

```
## Not run:
before <- Sys.time()
stepPenal<- StepPenal(Data=simData, lamda=1.5, w=0.3)
after <- Sys.time()
after-before

(varstepPenal<- stepPenal$coeffP)

## End(Not run)
```

`StepPenalL2` *Stepwise forward variable selection using penalized regression.*

Description

Stepwise forward variable selection based on the combination of L2 and L0 penalties. The optimization is done using the "BFGS" method in stats::optim

Usage

```
StepPenalL2(Data, lamda, w, standardize = TRUE)
```

Arguments

`Data`	should have the following structure: the first column must be the binary response variable y.
`lamda`	the tuning penalty parameter
`w`	the weight parameter for the sum (1-w)L0+ wL2
`standardize`	Logical flag for the predictors' standardization, prior to fitting the model. Default is standardize=TRUE

Details

lamda and w parameters need to be tuned by cross-Validation using stepPenal::tuneParam

Value

a list with the shrinked coefficients and the names of the selected variables, i.e those variables with an estimated coefficient different from zero.

See Also

`optim`

Examples

```
# use the StepPenal function on a simulated dataset, with given lamda and w.

set.seed(14)
beta    <- c(3, 2, -1.6, -1)
noise   <- 5
simData <- SimData(N=100, beta=beta, noise=noise, corr=TRUE)
## Not run:
before <- Sys.time()
stepPenalL2 <- StepPenalL2(Data=simData, lamda=1.5, w=0.6)
after <- Sys.time()
after-before

(varstepPenal<- stepPenalL2$coeffP)

## End(Not run)
```

`tuneParam` *Tune parameters w and lamda using the CL penalty*

Description

Does k-fold cross-validation with the function optimPenalLik and returns the values of lamda and w that maximize the area under the ROC.

Usage

```
tuneParam(Data, nfolds = nfolds, grid, algorithm = c("hjk", "QN"))
```

Arguments

`Data`	a data frame, as a first column should have the response variable y and the other columns the predictors
`nfolds`	the number of folds used for cross-validation. OBS! nfolds>=2
`grid`	a grid (data frame) with values of lamda and w that will be used for tuning to tune the model. It is created by expand.grid see example below
`algorithm`	choose between BFGS ("QN") and hjk (Hooke-Jeeves optimization free) to be used for optmization

Details

It supports the BFGS optimization method ('QN') from the optim stats function, the Hooke-Jeeves derivative-free minimization algorithm ('hjk') The value of lamda and w that yield the maximum AUC on the cross-validating data set is selected.

Value

A matrix with the following: the average (over folds) cross-validated AUC, the totalVariables selected on the training set, and the standard deviation of the AUC over the nfolds.

Examples

```
## Not run:
set.seed(14)
beta    <- c(3, 2, -1.6, -4)
noise   <- 5
simData <- SimData(N=100, beta=beta, noise=noise, corr=TRUE)

nfolds  <- 3
grid <- expand.grid(w = c( 0.3, 0.7),
                    lamda = c(1.5))

before <- Sys.time()
paramCV <- tuneParam(simData, nfolds, grid, algorithm=c("QN"))
(totalTime <- Sys.time() - before)


maxAUC    <- paramCV[which.max(paramCV$AUC),]$AUC
allmaxAUC <- paramCV[which(paramCV$AUC==maxAUC),] # checks if the value of AUC
# is unique; if is not unique then it will take the combination of lamda and
# w where lamda has the largest value- thus achieving higher sparsity

runQN   <- optimPenaLik(simData, lamda= allmaxAUC[nrow(allmaxAUC),]$lamda,
                        w= allmaxAUC[nrow(allmaxAUC),]$w,
                        algorithms=c("QN"))
(coefQN  <- runQN$varQN)


# check the robustness of the choice of lamda

runQN2   <- optimPenaLik(simData, lamda= allmaxAUC[1,]$lamda,
                        w= allmaxAUC[1,]$w,
                        algorithms=c("QN"))
(coefQN2  <- runQN2$varQN)


## End(Not run)
```

---

tuneParamCL2 *Tune parameters w and lamda using the CL2 penalty*

Description

Does k-fold cross-validation with the function optimPenalLikL2 and returns the values of lamda and w that maximize the area under the ROC.

Usage

```
tuneParamCL2(Data, nfolds = nfolds, grid, algorithm = c("QN"))
```

Arguments

Data	a data frame, as a first column should have the response variable y and the other columns the predictors
nfolds	the number of folds used for cross-validation. OBS! nfolds>=2
grid	a grid (data frame) with values of lamda and w that will be used for tuning to tune the model. It is created by expand.grid see example below
algorithm	choose between BFGS ("QN") and hjk (Hooke-Jeeves optimization free) to be used for optmization

Details

It supports the BFGS optimization method ('QN') from the optim stats function, the Hooke-Jeeves derivative-free minimization algorithm ('hjk'). The value of lamda and w that yield the maximum AUC on the cross-validating data set is selected. If more that one value of lamda nad w yield the same AUC, then the biggest values of lamda and w are choosen.

Value

A matrix with the following: the average (over folds) cross-validated AUC, the totalVariables selected on the training set, and the standard deviation of the AUC over the nfolds

Index