Package ‘penalizedSVM’

March 23, 2023

Type Package

Title Feature Selection SVM using Penalty Functions

Version 1.1.4

Date 2023-03-23

Depends e1071, mlegp, MASS

Imports corpcor, statmod, tgp

Author Natalia Becker, Wiebke Werft, Axel Benner

Maintainer Frederic Bertrand <frederic.bertrand@utt.fr>

Description

Support Vector Machine (SVM) classification with simultaneous feature selection using penalty functions is implemented. The smoothly clipped absolute deviation (SCAD), 'L1-norm', 'Elastic Net' ('L1-norm' and 'L2-norm') and 'Elastic SCAD' (SCAD and 'L2-norm') penalties are available. The tuning parameters can be found using either a fixed grid or a interval search.

License GPL (>= 2)

LazyLoad yes

RoxygenNote 6.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2023-03-23 15:10:02 UTC

R topics documented:

`penalizedSVM-package` *Feature Selection SVM using Penalty Functions*

Description

Feature Selection SVM using penalty functions. The smoothly clipped absolute deviation (SCAD) and L1-norm penalties are availible up to now. Other functions will be implemented in the near feature.

Details

Package:	penaltySVM
Type:	Package
Version:	1.1.3
Date:	2022-05-02
License:	GPL-2
LazyLoad:	yes

The main function is `svmfs`, `see the documentation file with examples`

Author(s)

Natalia Becker, Axel Benner, Wiebke Werft

Maintainer: Frederic Bertrand (<`frederic.bertrand@utt.fr`>)

References

Zhang, H. H., Ahn, J., Lin, X. and Park, C. (2006). *Gene selection using support vector machines with nonconvex penalty.* Bioinformatics, **22**, pp. 88-95.

Fung, G. and Mangasarian, O. L. (2004). *A feature selection newton method for support vector machine classification.* Computational Optimization and Applications Journal ,**28.2** , pp. 185-202.

`.plot.EPSGO.parms` *Plot Interval Search Plot Visited Points and the Q Values.*

Description

For interval search plot visited points and the Q values (=Ytrain) exclude: for D=1 make an additional plot: skip values for empty model, for example: Ytrain.exclude=10^16.

Usage

```
.plot.EPSGO.parms(Xtrain, Ytrain,bounds, Ytrain.exclude=10^16, plot.name=NULL )
```

Arguments

`Xtrain`	X points to train
`Ytrain`	Y points to train
`bounds`	bounds for parameters, see examples
`Ytrain.exclude`	If exclude for Ytrain exists, skip those points. Defaults to 10^16.
`plot.name`	Defaults to `NULL`

Details

The code source was adopted from MATLAB originals, special thanks to Holger Froehlich.

Value

None, only graphs are created.

Author(s)

Natalia Becker
<natalie_becker@gmx.de>

References

Froehlich, H. and Zell, A. (2005) "Effcient parameter selection for support vector machines in classification and regression via model-based global optimization" *In Proc. Int. Joint Conf. Neural Networks, 1431-1438* .

See Also

`svmfs`

Examples

```
seed <- 123

train<-sim.data(n = 200, ng = 100, nsg = 10, corr=FALSE, seed=seed )
print(str(train))

Q.func<- ".calc.scad"
```

```
bounds=t(data.frame(log2lambda1=c(-10, 10)))
colnames(bounds)<-c("lower", "upper")

print("start interval search")
# computation intensive;
# for demostration reasons only for the first 100 features
# and only for 10 iterations maxIter=10, default maxIter=700
system.time(fit<-EPSGO(Q.func, bounds=bounds, parms.coding="log2", fminlower=0,
 show='none', N=21,  maxevals=500,
 pdf.name=NULL,  seed=seed,
 verbose=FALSE,
 # Q.func specific parameters:
 x.svm=t(train$x)[,1:100], y.svm=train$y,
 inner.val.method="cv",
 cross.inner=5, maxIter=10 ))

print(paste("minimal 5-fold cv error:", fit$fmin, "by log2(lambda1)=", fit$xmin))

print(" all lambdas with the same minimum? ")
print(fit$ points.fmin)

print(paste(fit$neval, "visited points"))


print(" overview: over all visitied points in tuning parameter space
with corresponding cv errors")
print(data.frame(Xtrain=fit$Xtrain, cv.error=fit$Ytrain))

# create  3 plots om one screen:
# 1st plot: distribution of initial points in tuning parameter space
# 2nd plot: visited lambda points vs. cv errors
# 3rd plot: the same as the 2nd plot, Ytrain.exclude points are excluded.
#    The value cv.error = 10^16 stays for the cv error for an empty model !
.plot.EPSGO.parms (fit$Xtrain, fit$Ytrain,bound=bounds,
Ytrain.exclude=10^16, plot.name=NULL )
 # end of \donttest
```

EPSGO *Fits SVM with variable selection using penalties.*

Description

Fits SVM with feature selection using penalties SCAD and 1 norm.

Usage

```
EPSGO(Q.func, bounds,parms.coding="none", fminlower=0, flag.find.one.min =FALSE,
show=c("none", "final", "all"), N= NULL, maxevals = 500,
    pdf.name=NULL,  pdf.width=12,  pdf.height=12,   my.mfrow=c(1,1),
    verbose=TRUE, seed=123,  ...  )
```

Arguments

`Q.func` name of the function to be minimized.

`bounds` bounds for parameters, see examples

`parms.coding` parmeters coding: none or log2, default: none.

`fminlower` minimal value for the function Q.func, default is 0.

`flag.find.one.min` do you want to find one min value and stop? Default: FALSE

`show` show plots of DIRECT algorithm: none, final iteration, all iterations. Default: none

`N` define the number of start points, see details.

`maxevals` the maximum number of DIRECT function evaluations, default: 500.

`pdf.name` pdf name

`pdf.width` default 12

`pdf.height` default 12

`my.mfrow` default c(1,1)

`verbose` verbose? default TRUE.

`seed` seed

`...` additional argument(s)

Details

if the number of start points (N) is not defined by the user, it will be defined dependent on the dimensionality of the parameter space. N=10D+1, where D is the number of parameters, but for high dimensional parameter space with more than 6 dimensions, the initial set is restricted to 65. However for one-dimensional parameter space the N is set to 21 due to stability reasons.

The idea of EPSGO (Efficient Parameter Selection via Global Optimization): Beginning from an intial Latin hypercube sampling containing N starting points we train an Online GP, look for the point with the maximal expected improvement, sample there and update the Gaussian Process(GP). Thereby it is not so important that GP really correctly models the error surface of the SVM in parameter space, but that it can give a us information about potentially interesting points in parameter space where we should sample next. We continue with sampling points until some convergence criterion is met.

DIRECT is a sampling algorithm which requires no knowledge of the objective function gradient. Instead, the algorithm samples points in the domain, and uses the information it has obtained to decide where to search next. The DIRECT algorithm will globally converge to the maximal value of the objective function. The name DIRECT comes from the shortening of the phrase 'DIviding RECTangles', which describes the way the algorithm moves towards the optimum.

The code source was adopted from MATLAB originals, special thanks to Holger Froehlich.

Value

`fmin` minimal value of Q.func on the interval defined by bounds.

`xmin` coreesponding parameters for the minimum

`iter`	number of iterations
`neval`	number of visited points
`maxevals`	the maximum number of DIRECT function evaluations
`seed`	seed
`bounds`	bounds for parameters
`Q.func`	name of the function to be minimized.
`points.fmin`	the set of points with the same fmin
`Xtrain`	visited points
`Ytrain`	the output of Q.func at visited points Xtrain
`gp.seed`	seed for Gaussian Process
`model.list`	detailed information of the search process

Author(s)

Natalia Becker
`<natalie_becker@gmx.de>`

References

Froehlich, H. and Zell, A. (2005) "Effcient parameter selection for support vector machines in classification and regression via model-based global optimization" *In Proc. Int. Joint Conf. Neural Networks, 1431-1438* .

See Also

`svmfs`

Examples

```
seed <- 123

train<-sim.data(n = 200, ng = 100, nsg = 10, corr=FALSE, seed=seed )
print(str(train))

Q.func<- ".calc.scad"

bounds=t(data.frame(log2lambda1=c(-10, 10)))
colnames(bounds)<-c("lower", "upper")

print("start interval search")
# computation intensive;
# for demostration reasons only for the first 100 features
# and only for 10 iterations maxIter=10, default maxIter=700
system.time(fit<-EPSGO(Q.func, bounds=bounds, parms.coding="log2", fminlower=0,
 show='none', N=21,  maxevals=500,
 pdf.name=NULL,  seed=seed,
 verbose=FALSE,
 # Q.func specific parameters:
 x.svm=t(train$x)[,1:100], y.svm=train$y,
 inner.val.method="cv",
 cross.inner=5, maxIter=10 ))

print(paste("minimal 5-fold cv error:", fit$fmin, "by log2(lambda1)=", fit$xmin))

print(" all lambdas with the same minimum? ")
print(fit$ points.fmin)

print(paste(fit$neval, "visited points"))


print(" overview: over all visitied points in tuning parameter space
with corresponding cv errors")
print(data.frame(Xtrain=fit$Xtrain, cv.error=fit$Ytrain))

# create  3 plots om one screen:
# 1st plot: distribution of initial points in tuning parameter space
# 2nd plot: visited lambda points vs. cv errors
# 3rd plot: the same as the 2nd plot, Ytrain.exclude points are excluded.
#    The value cv.error = 10^16 stays for the cv error for an empty model !
.plot.EPSGO.parms (fit$Xtrain, fit$Ytrain,bound=bounds,
Ytrain.exclude=10^16, plot.name=NULL )
 # end of \donttest
```

`findgacv.scad` *Calculate Generalized Approximate Cross Validation Error Estimation for SCAD SVM model*

Description

calculate generalized approximate cross validation error (GACV) estimation for SCAD SVM model

Usage

```
findgacv.scad(y, model)
```

Arguments

`y`	vector of class labels (only for 2 classes)
`model`	list, describing SCAD SVM model, produced by function scadsvc

Value

returns the GACV value

Author(s)

Natalia Becker
<natalie_becker@gmx.de>

References

Zhang, H. H., Ahn, J., Lin, X. and Park, C. (2006). *Gene selection using support vector machines with nonconvex penalty.* Bioinformatics, **22**, pp. 88-95.

Wahba G., Lin, Y. and Zhang, H. (2000). *GACV for support vector machines, or, another way to look at margin-like quantities, in A. J. Smola, P. Bartlett, B. Schoelkopf and D. Schurmans (eds)*, Advances in Large Margin Classifiers, MIT Press, pp. 297-309.

See Also

`scadsvc`, `predict.penSVM`, `sim.data`

Examples

```
# simulate data
train<-sim.data(n = 200, ng = 100, nsg = 10, corr=FALSE, seed=12)
print(str(train))

# train data
ff <- scadsvc(as.matrix(t(train$x)), y=train$y, lambda=0.01)
print(str(ff))

# estimate gacv error
(gacv<- findgacv.scad(train$y, model=ff))
```

`lpsvm` *Fit L1-norm SVM*

Description

SVM with variable selection (clone selection) using L1-norm penalty. (a fast Newton algorithm NLPSVM from Fung and Mangasarian)

Usage

```
lpsvm(A, d, k = 5, nu = 0, output = 1, delta = 10^-3, epsi = 10^-4,
seed = 123, maxIter=700)
```

Arguments

`A`	n-by-d data matrix to train (n chips/patients, d clones/genes).
`d`	vector of class labels -1 or 1's (for n chips/patiens).
`k`	k-fold for cv, default k=5.
`nu`	weighted parameter, 1 - easy estimation, 0 - hard estimation, any other value - used as nu by the algorithm. Default : 0.
`output`	0 - no output, 1 - produce output, default is 0.
`delta`	some small value, default: 10^-3.
`epsi`	tuning parameter.
`seed`	seed.
`maxIter`	maximal iterations, default: 700.

Details

k: k-fold for cv, is a way to divide the data set into test and training set.
if k = 0: simply run the algorithm without any correctness calculation, this is the default.
if k = 1: run the algorithm and calculate correctness on the whole data set.
if k = any value less than the number of rows in the data set: divide up the data set into test and training using k-fold method.
if k = number of rows in the data set: use the 'leave one out' (loo) method

Value

a list of

`w`	coefficients of the hyperplane
`b`	intercept of the hyperplane
`xind`	the index of the selected features (genes) in the data matrix.
`epsi`	optimal tuning parameter epsilon
`iter`	number of iterations
`k`	k-fold for cv
`trainCorr`	for cv: average train correctness
`testCorr`	for cv: average test correctness
`nu`	weighted parameter

Note

Adapted from MATLAB code http://www.cs.wisc.edu/dmi/svm/lpsvm/

Author(s)

Natalia Becker

References

Fung, G. and Mangasarian, O. L. (2004). A feature selection newton method for support vector machine classification. *Computational Optimization and Applications Journal 28(2) pp. 185-202*.

See Also

sim.data

Examples

```
set.seed(123)
train<-sim.data(n = 20, ng = 100, nsg = 10, corr=FALSE, seed=12)
print(str(train))

# train data
model <- lpsvm(A=t(train$x), d=train$y, k=5, nu=0,output=0, delta=10^-3, epsi=0.001, seed=12)
print(model)
```

predict *Predict Method for Feature Selection SVM*

Description

This function predicts values based upon a model trained by svm. If class assigment is provided, confusion table, missclassification table, sensitivity and specificity are calculated.

Usage

```
## S3 method for class 'penSVM'
predict(object, newdata, newdata.labels = NULL, labels.universe=NULL, ...)
```

Arguments

`object` Object of class "penSVM", created by 'svmfs'

`newdata` A matrix containing the new input data, samples in rows, features in columns

`newdata.labels` optional, new data class labels

`labels.universe` important for models produced by loocv: all possible labels in the particular data set

`...` additional argument(s)

Value

returns a list of prediction values for classes

`pred.class`	predicted class
`tab`	confusion table
`error`	missclassification error
`sensitivity`	sensitivity
`specificity`	specificity

Author(s)

Natalia Becker

See Also

`svm`, `svmfs`

Examples

```
seed<- 123
train<-sim.data(n = 200, ng = 100, nsg = 10, corr=FALSE, seed=seed )
print(str(train))

#train standard svm
my.svm<-svm(x=t(train$x), y=train$y, kernel="linear")

# test with other data
test<- sim.data(n = 200, ng = 100, nsg = 10, seed=(seed+1) )

# Check accuracy standard SVM
my.pred <-ifelse( predict(my.svm, t(test$x)) >0,1,-1)
# Check accuracy:
table(my.pred, test$y)

## Not run: # define set values of tuning parameter lambda1 for SCAD
lambda1.scad <- c (seq(0.01 ,0.05, .01),  seq(0.1,0.5, 0.2), 1 )
# for presentation don't check  all lambdas : time consuming!
# computation intensive; for demostration reasons only for the first 100 features
# and only for 10 Iterations maxIter=10, default maxIter=700

system.time(fit.scad<- svmfs(x=t(train$x)[,1:100],y=train$y, fs.method="scad", cross.outer= 0,
grid.search = "discrete",  lambda1.set=lambda1.scad[1:3], show="none",
parms.coding = "none", maxIter=10,
inner.val.method = "cv", cross.inner= 5, seed=seed, verbose=FALSE))


# SCAD
test.error.scad<-predict(fit.scad, newdata=t(test$x)[,1:100],newdata.labels=test$y )
 # Check accuracy SCAD SVM
```

```
print(test.error.scad$tab)

## End(Not run)

##########################################
## analog for 1-norm SVM
#epsi.set<-vector(); for (num in (1:9)) epsi.set<-sort(c(epsi.set, c(num*10^seq(-5, -1, 1 ))) )
#lambda1.1norm <-  epsi.set[c(3,5)] # 2 params
#
## train 1norm SVM
# norm1.fix<- svmfs(t(train$x), y=train$y, fs.method=”1norm”,
# cross.outer= 0, grid.search = "discrete",
# lambda1.set=lambda1.1norm, show=”none”,
# parms.coding = ”none”,
# maxIter = 700, inner.val.method = ”cv”, cross.inner= 5,
# seed=seed, verbose=FALSE )
#
# print(norm1.fix)
#
## L1-norm SVM
#test.error.1norm<-predict(norm1.fix, newdata=t(test$x),newdata.labels=test$y )
# # Check accuracy L1-norm SVM
#print(test.error.1norm$tab)
```

print *Print Function for FS SVM*

Description

Print Function for FS SVM

Usage

```
## S3 method for class 'penSVM'
print(x,...)
```

Arguments

x model trained by scad or 1norm svm of class PenSVM

... additional argument(s)

Author(s)

Natalia Becker

See Also

svm, svmfs

Examples

```
seed<- 123
train<-sim.data(n = 20, ng = 100, nsg = 10, corr=FALSE, seed=seed )
print(str(train))

# for presentation don't check  all lambdas : time consuming!
model <- scadsvc(as.matrix(t(train$x)), y=train$y, lambda=0.05)
print(str(model))

print(model)
```

`scadsvc` *Fit SCAD SVM model*

Description

SVM with variable selection (clone selection) using SCAD penalty.

Usage

```
scadsvc(lambda1 = 0.01, x, y, a = 3.7, tol= 10^(-4), class.weights= NULL,
 seed=123, maxIter=700, verbose=TRUE)
```

Arguments

`lambda1` tuning parameter in SCAD function (default : 0.01)

`x` n-by-d data matrix to train (n chips/patients, d clones/genes)

`y` vector of class labels -1 or 1\’s (for n chips/patiens)

`a` tuning parameter in scad function (default: 3.7)

`tol` the cut-off value to be taken as 0

`class.weights` a named vector of weights for the different classes, used for asymetric class sizes. Not all factor levels have to be supplied (default weight: 1). All components have to be named. (default: NULL)

`seed` seed

`maxIter` maximal iteration, default: 700

`verbose` verbose, default: TRUE

Details

Adopted from Matlab code: http://www4.stat.ncsu.edu/~hzhang/software.html

Value

`w` coefficients of the hyperplane.

`b` intercept of the hyperplane.

`xind` the index of the selected features (genes) in the data matrix.

`xqx` internal calculations product $xqx = 0.5 * x1 * inv_Q * t(x1)$, see code for more details.

`fitted` fit of hyperplane f(x) for all _training_ samples with reduced set of features.

`index` the index of the resulting support vectors in the data matrix.

`type` type of svm, from svm function.

`lambda1` optimal lambda1.

`gacv` corresponding gacv.

`iter` nuber of iterations.

Author(s)

Axel Benner

References

Zhang, H. H., Ahn, J., Lin, X. and Park, C. (2006). *Gene selection using support vector machines with nonconvex penalty. Bioinformatics, **22**, pp. 88-95.*

See Also

`findgacv.scad`, `predict.penSVM`, `sim.data`

Examples

```
# simulate data
train<-sim.data(n = 200, ng = 100, nsg = 10, corr=FALSE, seed=12)
print(str(train))

# train data
model <- scadsvc(as.matrix(t(train$x)), y=train$y, lambda=0.01)
print(str(model))

print(model)
```

`sim.data` *Simulation of microarray data*

Description

Simulation of 'n' samples. Each sample has 'sg' genes, only 'nsg' of them are called significant and have influence on class labels. All other '(ng - nsg)' genes are called ballanced. All gene ratios are drawn from a multivariate normal distribution. There is a posibility to create blocks of highly correlated genes.

Usage

```
sim.data(n = 256, ng = 1000, nsg = 100,
 p.n.ratio = 0.5,
 sg.pos.factor= 1, sg.neg.factor= -1,
 # correlation info:
 corr = FALSE, corr.factor = 0.8,
 # block info:
 blocks = FALSE, n.blocks = 6, nsg.block = 1, ng.block = 5,
 seed = 123, ...)
```

Arguments

`n`	number of samples, logistic regression works well if $n > 200$!
`ng`	number of genes
`nsg`	number of significant genes
`p.n.ratio`	ratio between positive and negative significant genes (default 0.5)
`sg.pos.factor`	impact factor of **positive** significant genes on the classifaction, default: 1
`sg.neg.factor`	impact factor of **negative** significant genes on the classifaction,default: -1
`corr`	are the genes correalted to each other? (default FALSE). see Details
`corr.factor`	correlation factorfor genes, between 0 and 1 (default 0.8)
`blocks`	are blocks of highly correlated genes are allowed? (default FALSE)
`n.blocks`	number of blocks
`nsg.block`	number of significant genes per block
`ng.block`	number of genes per block
`seed`	seed
`...`	additional argument(s)

Details

If no blockes (n.blocks=0 or blocks=FALSE) are defined and corr=TRUE create covarance matrix for all genes! with decrease of correlation : $cov(i, j) = cov(j, i) = corr.factor^(i - j)$

Value

`x` matrix of simulated data. Genes in rows and samples in columns

`y` named vector of class labels

`seed` seed

Author(s)

Wiebke Werft, Natalia Becker

See Also

`mvrnorm`

Examples

```
my.seed<-123

# 1. simulate 20 samples, with 100 genes in each. Only the first two genes
# have an impact on the class labels.
# All genes are assumed to be i.i.d.
train<-sim.data(n = 20, ng = 100, nsg = 3, corr=FALSE, seed=my.seed )
print(str(train))

# 2. change the proportion between positive and negative significant genes
#(from 0.5 to 0.8)
train<-sim.data(n = 20, ng = 100, nsg = 10, p.n.ratio = 0.8,  seed=my.seed )
rownames(train$x)[1:15]
# [1] ”pos1” ”pos2” ”pos3” ”pos4” ”pos5” ”pos6” ”pos7” ”pos8”
# [2] ”neg1” ”neg2” ”bal1” ”bal2” ”bal3” ”bal4” ”bal5”

# 3. assume to have correlation for positive significant genes,
# negative significant genes and 'balanced' genes separatly.
train<-sim.data(n = 20, ng = 100, nsg = 10, corr=TRUE, seed=my.seed )
#cor(t(train$x[1:15,]))

# 4. add 6 blocks of 5 genes each and only one significant gene per block.
# all genes in the block are correlated with constant correlation factor
#  corr.factor=0.8
train<-sim.data(n = 20, ng = 100, nsg = 6, corr=TRUE, corr.factor=0.8,
 blocks=TRUE, n.blocks=6, nsg.block=1, ng.block=5, seed=my.seed )
print(str(train))
# first block
#cor(t(train$x[1:5,]))
# second block
#cor(t(train$x[6:10,]))
```

sortmat *Sort matrix or data frame*

Description

A useful function for ranking. Sort matrix or dataframe 'Mat', by column(s) 'Sort' in decrising or increasing order.

Usage

```
sortmat (Mat, Sort, decreasing=FALSE)
```

Arguments

`Mat`	a matrix or a data frame
`Sort`	Sort is a number !
`decreasing`	in decreasing order? default: FALSE

Value

sorted matrix or data frame

Author(s)

found in world wide web: http://tolstoy.newcastle.edu.au/R/help/99b/0668.html

Examples

```
m <- matrix(c(9:5, c(1, 4, 3, 3, 5), c(1, 2, 4, 3, 5)), ncol = 3, byrow = FALSE)

print( m)
#    [,1] [,2] [,3]
#[1,]    9    1    1
#[2,]    8    4    2
#[3,]    7    3    4
#[4,]    6    3    3
#[5,]    5    5    5

# sort first according to the second column then if equal according to the third column
print(m1 <- sortmat(Mat = m, Sort = c(2, 3)))
#     [,1] [,2] [,3]
#[1,]    9    1    1
#[2,]    6    3    3
#[3,]    7    3    4
#[4,]    8    4    2
#[5,]    5    5    5
```

```
# sort first according to the third (!)  column then if equal according
# to the second column
print(m2 <- sortmat(Mat = m, Sort = c(3, 2)))
#     [,1] [,2] [,3]
#[1,]    9    1    1
#[2,]    8    4    2
#[3,]    6    3    3
#[4,]    7    3    4
#[5,]    5    5    5

# Note m1 and m2 are not equal!!!!
all(m1==m2) #FALSE


# in decreasing order
print(m3 <- sortmat(Mat = m, Sort = c(2, 3), decreasing=TRUE))
#     [,1] [,2] [,3]
#[1,]    5    5    5
#[2,]    8    4    2
#[3,]    7    3    4
#[4,]    6    3    3
#[5,]    9    1    1
```

---

svmfs *Fits SVM with variable selection using penalties.*

Description

Fits SVM with variable selection (clone selection) using penalties SCAD, L1 norm, Elastic Net (L1 + L2 norms) and ELastic SCAD (SCAD + L1 norm). Additionally tuning parameter search is presented by two approcaches: fixed grid or interval search. NOTE: The name of the function has been changed: svmfs instead of svm.fs!

Usage

```
## Default S3 method:
svmfs(x,y,
fs.method = c("scad", "1norm", "scad+L2", "DrHSVM"),
grid.search=c("interval","discrete"),
lambda1.set=NULL,
lambda2.set=NULL,
bounds=NULL,
parms.coding= c("log2","none"),
maxevals=500,
inner.val.method = c("cv", "gacv"),
cross.inner= 5,
show= c("none", "final"),
calc.class.weights=FALSE,
class.weights=NULL,
seed=123,
maxIter=700,
verbose=TRUE,
...)
```

Arguments

`x` input matrix with genes in columns and samples in rows!

`y` numerical vector of class labels, -1 , 1

`fs.method` feature selection method. Availible 'scad', '1norm' for 1-norm, "DrHSVM" for Elastic Net and "scad+L2" for Elastic SCAD

`grid.search` chose the search method for tuning lambda1,2: 'interval' or 'discrete', default: 'interval'

`lambda1.set` for fixed grid search: fixed grid for lambda1, default: NULL

`lambda2.set` for fixed grid search: fixed grid for lambda2, default: NULL

`bounds` for interval grid search: fixed grid for lambda2, default: NULL

`parms.coding` for interval grid search: parms.coding: none or log2 , default: log2

`maxevals` the maximum number of DIRECT function evaluations, default: 500.

`calc.class.weights`
calculate class.weights for SVM, default: FALSE

`class.weights` a named vector of weights for the different classes, used for asymetric class sizes. Not all factor levels have to be supplied (default weight: 1). All components have to be named.

`inner.val.method`
method for the inner validation: cross validation, gacv , default cv

`cross.inner` 'cross.inner'-fold cv, default: 5

`show` for interval search: show plots of DIRECT algorithm: none, final iteration, all iterations. Default: none

`seed` seed

`maxIter` maximal iteration, default: 700

`verbose` verbose?, default: TRUE

`...` additional argument(s)

Details

The goodness of the model is highly correlated with the choice of tuning parameter lambda. Therefore the model is trained with different lambdas and the best model with optimal tuning parameter is used in futher analysises. For very small lamdas is recomended to use maxIter, otherweise the algorithms is slow or might not converge.

The Feature Selection methods are using different techniques for finding optimal tunung parameters By SCAD SVM Generalized approximate cross validation (gacv) error is calculated for each predefined tuning parameter.

By L1-norm SVM the cross validation (default 5-fold) missclassification error is calculated for each lambda. After training and cross validation, the optimal lambda with minimal missclassification error is choosen, and a final model with optimal lambda is created for the whole data set.

Value

`classes` vector of class labels as input 'y'

`sample.names` sample names

`class.method` feature selection method

`seed` seed

`model` final model

- w - coefficients of the hyperplane
- b - intercept of the hyperplane
- xind - the index of the selected features (genes) in the data matrix.
- index - the index of the resulting support vectors in the data matrix.
- type - type of svm, from svm function
- lam.opt - optimal lambda
- gacv - corresponding gacv

Author(s)

Natalia Becker
<natalie_becker@gmx.de>

References

Becker, N., Werft, W., Toedt, G., Lichter, P. and Benner, A.(2009) PenalizedSVM: a R-package for feature selection SVM classification, Bioinformatics, 25(13),p 1711-1712

See Also

`predict.penSVM`, `svm` (in package **e1071**)

Examples

```




seed<- 123

train<-sim.data(n = 200, ng = 100, nsg = 10, corr=FALSE, seed=seed )
print(str(train))


### Fixed grid ####

# train SCAD SVM ####################
# define set values of tuning parameter lambda1 for SCAD
lambda1.scad <- c (seq(0.01 ,0.05, .01),  seq(0.1,0.5, 0.2), 1 )
# for presentation don't check  all lambdas : time consuming!
lambda1.scad<-lambda1.scad[2:3]
```

```
#
# train SCAD SVM

# computation intensive; for demostration reasons only for the first 100 features
# and only for 10 Iterations maxIter=10, default maxIter=700
system.time(scad.fix<- svmfs(t(train$x)[,1:100], y=train$y, fs.method="scad",
   cross.outer= 0, grid.search = "discrete",
   lambda1.set=lambda1.scad,
   parms.coding = "none", show="none",
   maxIter = 10, inner.val.method = "cv", cross.inner= 5,
   seed=seed, verbose=FALSE)  )

print(scad.fix)

# train 1NORM SVM  ################
# define set values of tuning parameter lambda1 for 1norm
#epsi.set<-vector(); for (num in (1:9)) epsi.set<-sort(c(epsi.set,
#    c(num*10^seq(-5, -1, 1 ))) )
## for presentation don't check  all lambdas : time consuming!
#lambda1.1norm <-  epsi.set[c(3,5)] # 2 params
#
### train 1norm SVM
## time consuming: for presentation only for the first 100 features
#norm1.fix<- svmfs(t(train$x)[,1:100], y=train$y, fs.method="1norm",
# cross.outer= 0, grid.search = "discrete",
# lambda1.set=lambda1.1norm,
# parms.coding = "none", show="none",
# maxIter = 700, inner.val.method = "cv", cross.inner= 5,
# seed=seed, verbose=FALSE )
#
# print(norm1.fix)

### Interval  search  ####


seed <- 123

train<-sim.data(n = 200, ng = 100, nsg = 10, corr=FALSE, seed=seed )
print(str(train))


test<-sim.data(n = 200, ng = 100, nsg = 10, corr=FALSE, seed=seed+1 )
print(str(test))


bounds=t(data.frame(log2lambda1=c(-10, 10)))
colnames(bounds)<-c("lower", "upper")

# computation intensive; for demostration reasons only for the first 100 features
# and only for 10 Iterations maxIter=10, default maxIter=700
print("start interval search")
system.time( scad<- svmfs(t(train$x)[,1:100], y=train$y,
 fs.method="scad", bounds=bounds,
 cross.outer= 0, grid.search = "interval",  maxIter = 10,
 inner.val.method = "cv", cross.inner= 5, maxevals=500,
 seed=seed, parms.coding = "log2", show="none", verbose=FALSE ) )
print("scad final model")
print(str(scad$model))

(scad.5cv.test<-predict.penSVM(scad, t(test$x)[,1:100], newdata.labels=test$y)   )


print(paste("minimal 5-fold cv error:", scad$model$fit.info$fmin,
"by log2(lambda1)=", scad$model$fit.info$xmin))

print(" all lambdas with the same minimum? ")
print(scad$model$fit.info$ points.fmin)

print(paste(scad$model$fit.info$neval, "visited points"))


print(" overview: over all visitied points in tuning parameter space
with corresponding cv errors")
print(data.frame(Xtrain=scad$model$fit.info$Xtrain,
cv.error=scad$model$fit.info$Ytrain))
#

# create  3 plots on one screen:
# 1st plot: distribution of initial points in tuning parameter space
# 2nd plot: visited lambda points vs. cv errors
# 3rd plot: the same as the 2nd plot, Ytrain.exclude points are excluded.
# The value cv.error = 10^16 stays for the cv error for an empty model !
.plot.EPSGO.parms (scad$model$fit.info$Xtrain, scad$model$fit.info$Ytrain,
bound=bounds, Ytrain.exclude=10^16, plot.name=NULL )

 # end of \donttest
```

Index