Package 'pliable'

February 20, 2019

Type Package

Title The Pliable Lasso

Version 1.1

Date 2019-02-11

Author Robert Tibshirani, Jerome Friedman

Maintainer Robert Tibshirani <tibs@stanford.edu>

Depends class, glmnet

License GPL-3

Suggests knitr, rmarkdown

Description
Fits a pliable lasso model. For details see Tibshirani and Friedman (2018) <arXiv:1712.00484>.

URL https://arxiv.org/abs/1712.00484

VignetteBuilder knitr

NeedsCompilation yes

Repository CRAN

Date/Publication 2019-02-20 09:50:03 UTC

RoxygenNote 6.1.1

R topics documented:

`coef.pliable` *Return estimated coefficients from a fitted pliable model.*

Description

Return estimated coefficients from a fitted pliable model.

Usage

```
## S3 method for class 'pliable'
coef(object, lambda = NULL, ...)
```

Arguments

`object`	object returned from a call to pliable
`lambda`	values of lambda at whcih predictions are desired. If NULL (default), the path of lambda values from the fitted model. are used. If lambda is not NULL, the predictions are made at the closest values to lambda in the lambda path from the fitted model;
`...`	Further arguments (not used)
	@return estimated parameters

`cv.pliable` *Carries out cross-validation for a pliable lasso model over a path of regularization values*

Description

Carries out cross-validation for a pliable lasso model over a path of regularization values

Usage

```
cv.pliable(fit, x, z, y, nfolds = 10, foldid = NULL, keep = F,
  type.measure = c("deviance", "class"), verbose = TRUE)
```

Arguments

`fit`	object returned by the pliable function
`x`	n by p matrix of predictors
`z`	n by nz matrix of modifying variables. Note that z may be observed or the predictions from a supervised learning algorithm that predicts z from validation set features x. IMPORTANT: in the latter case, the z matrix must be pre-computed before calling cv.pliable, using the same CV folds used by cv.pliable. See the example below

y	n-vector of responses. All variables are centered in the function.
nfolds	number of cross-validation folds
foldid	vector with values in 1:K, indicating folds for K-fold CV. Default NULL
keep	Should pre-validated fits be returned? Default FALSE
type.measure	Error measure for cross-validation: "deviance" (mse) for gaussian family, "deviance" or "class" (misclassification error) for binomial family
verbose	Should information be printed along the way? Default FALSE

Value

predicted values #'

Examples

```
# Train and cross-validate a pliable lasso model- Gaussian case
n = 20 ; p = 3 ;nz=3
x = matrix(rnorm(n*p), n, p)
mx=colMeans(x)
sx=sqrt(apply(x,2,var))
x=scale(x,mx,sx)
z =matrix(rnorm(n*nz),n,nz)
mz=colMeans(z)
sz=sqrt(apply(z,2,var))
z=scale(z,mz,sz)
y =4*x[,1] +5*x[,1]*z[,3]+ 3*rnorm(n)
fit = pliable(x,z,y)

cvfit=cv.pliable(fit,x,z,y,nfolds=5)
  plot(cvfit)

# Example of categorical z with 4 levels
n = 20; p = 3 ;nz=3
x = matrix(rnorm(n*p), n, p)
 mx=colMeans(x)
sx=sqrt(apply(x,2,var))
x=scale(x,mx,sx)
z =sample(1:4,size=n,replace=T)
zi=model.matrix(~as.factor(z)-1)
y = x[,1] +x[,1]*zi[,3]-2*x[,1]*zi[,4]+rnorm(n)
fit = pliable(x,zi,y)

# Train and cross-validate a pliable lasso model- Binomial case
n = 20; p = 3 ;nz=3
x = matrix(rnorm(n*p), n, p)
mx=colMeans(x)
sx=sqrt(apply(x,2,var))
x=scale(x,mx,sx)
z =matrix(rnorm(n*nz),n,nz)
mz=colMeans(z)
sz=sqrt(apply(z,2,var))
```

```
z=scale(z,mz,sz)
y =4*x[,1] +5*x[,1]*z[,3]+ 3*rnorm(n)
y= 1*(y>0)
fit = pliable(x,z,y)

cvfit=cv.pliable(fit,x,z,y,type.measure="class", nfolds=5)
  plot(cvfit)


 # Example where z is not observed in the test set,
   #   but predicted from a supervised learning algorithm
# NOT RUN

#library(glmnet)
# n = 20 ; p = 4 ;nz=3
# x = matrix(rnorm(n*p), n, p)
# mx=colMeans(x)
# sx=sqrt(apply(x,2,var))
# x=scale(x,mx,sx)
# z =matrix(rnorm(n*nz),n,nz)
# mz=colMeans(z)
# sz=sqrt(apply(z,2,var))
# z=scale(z,mz,sz)
#y =4*x[,1] +5*x[,1]*z[,3]+ 3*rnorm(n)
#fit = pliable(x,z,y)

 #nfolds=5
#  set.seed(3321)
#foldid = sample(rep(seq(nfolds), length = n))

# predict z  from x; here we use glmnet, but any other supervised learning procedure
  #  could be used
#zhat=matrix(NA,n,ncol(z))
#for(ii in 1:nfolds){
#  zfit=cv.glmnet(x[foldid!=ii,],z[foldid!=ii,],family="mgaussian")
#  zhat[foldid==ii,]=predict(zfit,x[foldid==ii,],s=zfit$lambda.min)
#}

#NOTE that the same foldid vector must be passed to cv.pliable
#cvfit=cv.pliable(fit,x,zhat,y,foldid=foldid)
#plot(cvfit)
#
```

`pliable` *Fit a pliable lasso model over a path of regularization values*

Description

Fit a pliable lasso model over a path of regularization values

Usage

```
pliable(x, z, y, lambda = NULL, nlambda = 50, alpha = 0.5,
  family = c("gaussian", "binomial"), lambda.min.ratio = NULL,
  w = NULL, thr = 1e-05, tt = NULL, penalty.factor = rep(1,
  ncol(x)), maxit = 10000, mxthit = 100, mxkbt = 100, mxth = 1000,
  kpmax = 1e+05, kthmax = 1e+05, verbose = FALSE, maxinter = 500,
  zlinear = TRUE, screen = NULL)
```

Arguments

x	n by p matrix of predictors
z	n by nz matrix of modifying variables. The elements of z may represent quantitative or categorical variables, or a mixture of the two. Categorical varables should be coded by 0-1 dummy variables: for a k-level variable, one can use either k or k-1 dummy variables.
y	n-vector of responses. The x and z variables are centered in the function. We recommmend that x and z also be standardized before the call via x<-scale(x,T,T)/sqrt(nrow(x)-1); z<-scale(z,T,T)/sqrt(nrow(z)-1)
lambda	decreasing sequence of lam values desired. Default NULL. If NULL, computed in function
nlambda	number of lambda values desired (default 50).
alpha	mixing parameter- default 0.5
family	response type- either "gaussian", "binomial". In the binomial case, y should be 0s and 1s. family "cox" (Prop Hazards model for right-censored data) will be implemented if there is popular demand
lambda.min.ratio	the smallest value for lambda , as a fraction of lambda.max, the (data derived) entry value (i.e. the smallest value for which all coefficients are zero). Default is 0.001 if n>p, and 0.01 if n< p. Along with "screen" (see below), this parameter is a main parameter controlling the runtime of the algorithm, With a smaller lambda.min.ratio, more interactions will typically be entered. If the algorithm stalls or stops near the end of the path, trying increasing lambda.min.ratio On the other hand, if a more complex fit is desired, or the cross-validation curve from cv.pliable is still going down at the end of the path, consider decreasing lambda.min.ratio
w	n-vector of observation wts (Default: all ones). Not currently used with Cox family.
thr	convergence threshold for estimation of beta and joint estimation of (beta,theta). Default 1e-5
tt	initial factor for Generalized grad algorithm for joint estimation of main effects and interaction parameters- default 0.1/mean(x^2)
penalty.factor	vector of penalty factors, one for each X predictors. Default: a vector of ones
maxit	maximum number of iterations in loop for one lambda. Default 10000
mxthit	maximum number of theta solution iterations. Default 1000

mxkbt	maximum number of backtracking iterations. Default 100
mxth	maximum internal theta storage. Default 1000
kpmax	maximum dimension of main effect storage. Default 100000
kthmax	maximum dimension of interaction storage. Default 100000
verbose	Should information be printed along the way? Default FALSE
maxinter	Maximum number of interactions allowed in the model. Default 500
zlinear	Should an (unregularized) linear term for z be included in the model? Default TRUE.
screen	Screening quantile for features: for example a value screen=0.7 means that we keep only those features with univariate t-stats above the 0.7 quantile of all features. Default is NULL, meaning that all features are kept

Details

This function fits a pliable lasso model over a sequence of lambda (regularization) values. The inputs are a feature matrix x, response vector y and a matrix of modifying variables z. The pliable lasso is a generalization of the lasso in which the weights multiplying the features x are allowed to vary as a function of a set of modifying variables z. The model has the form:

yhat=beta0+ z%*% betaz+ rowSums(x[,j]*beta[j] + x[,j]*z%*% theta[j,])

The parameters are beta0 (scalar), betaz (nz-vector), beta (p-vector) and theta (p by nz). The (convex) optimization problem is

minimize_(beta0,betaz,beta,theta) (1/(2n))* sum_i ((y_i-yhat_i)^2) +(1-alpha)*lambda* sum_j (||(beta_j,theta_j)||_2 +||theta_j)|| +alpha*lambda* sum_j||theta_j||_1

A blockwise coordinate descent procedure is employed for the optimization.

Value

A trained pliable object with the following components:

param: values of lambda used

beta: matrix of estimated beta coefficients ncol(x) by length(lambda).

theta: array of estimated theta coefficients ncol(x) by ncol(z) by length(lambda).

betaz: estimated (main effect) coefficients for z ncol(z) by length(lambda).

xbar: rowMeans of x

zbar: rowMeans of z

w: observation weights used

args: list of arguments in the original call

nulldev: null deviance for model

dev.ratio: deviance/null.dev for each model in path

nbeta: number of nonzero betas for each model in path

nbeta.with.int: number of betas with some nonzero theta values, for each model in path

ntheta: number of nonzero theta for each model in path

df: rough degrees of freedom for each model in path (=nbeta)

istor: for internal use

fstor: for internal use

thstor: for internal use

jerr: for internal use

call: the call made to pliable

See Also

cv.pliable, predict.pliable, plot.pliable, plot.cv.pliable

Examples

```
# Train a pliable lasso model- Gaussian case
#  Generate some data
set.seed(9334)
n = 20; p = 3 ;nz=3
x = matrix(rnorm(n*p), n, p)
mx=colMeans(x)
sx=sqrt(apply(x,2,var))
x=scale(x,mx,sx)
z =matrix(rnorm(n*nz),n,nz)
mz=colMeans(z)
sz=sqrt(apply(z,2,var))
z=scale(z,mz,sz)
y =4*x[,1] +5*x[,1]*z[,3]+ 3*rnorm(n)
 # Fit model
 fit = pliable(x,z,y)
 fit   #examine results
plot(fit)  #plot path

#Estimate main tuning parameter lambda by cross-validation
 #NOT RUN
 # cvfit=cv.pliable(fit,x,z,y,nfolds=5) # returns lambda.min and lambda.1se,
    #  the minimizing and 1se rules
 # plot(cvfit)

# Predict using the fitted model
#ntest=500
#xtest = matrix(rnorm(ntest*p),ntest,p)
#xtest=scale(xtest,mx,sx)
#ztest =matrix(rnorm(ntest*nz),ntest,nz)
#ztest=scale(ztest,mz,sz)
#ytest = 4*xtest[,1] +5*xtest[,1]*ztest[,3]+ 3*rnorm(ntest)
#pred= predict(fit,xtest,ztest,lambda=cvfit$lambda.min)
#plot(ytest,pred)

#Binary outcome example
n = 20 ; p = 3 ;nz=3
x = matrix(rnorm(n*p), n, p)
```

```
mx=colMeans(x)
sx=sqrt(apply(x,2,var))
x=scale(x,mx,sx)
z =matrix(rnorm(n*nz),n,nz)
mz=colMeans(z)
sz=sqrt(apply(z,2,var))
z=scale(z,mz,sz)
y =4*x[,1] +5*x[,1]*z[,3]+ 3*rnorm(n)
y=1*(y>0)
 fit = pliable(x,z,y,family="binomial")



# Example where z is not observed in the test set, but predicted
 #        from a supervised learning algorithm


n = 20; p = 3 ;nz=3
x = matrix(rnorm(n*p), n, p)
mx=colMeans(x)
sx=sqrt(apply(x,2,var))
x=scale(x,mx,sx)
z =matrix(rnorm(n*nz),n,nz)
mz=colMeans(z)
sz=sqrt(apply(z,2,var))
z=scale(z,mz,sz)
y =4*x[,1] +5*x[,1]*z[,3]+ 3*rnorm(n)

fit = pliable(x,z,y)
# predict z  from x; here we use glmnet, but any other supervised method can be used


zfit=cv.glmnet(x,z,family="mgaussian")

# Predict using the fitted model
# ntest=100
#xtest =matrix(rnorm(ntest*nz),ntest,p)
#xtest=scale(xtest,mx,sx)
#ztest =predict(zfit,xtest,s=cvfit$lambda.min)[,,1]
#ytest = 4*xtest[,1] +5*xtest[,1]*ztest[,3]+ 3*rnorm(ntest)

#pred= predict(fit,xtest,ztest)
#plot(ytest,pred[,27])  # Just used 27th path member, for illustration; see cv.pliable for
#  details of how to cross-validate in this setting
```

`predict.pliable` *Compute predicted values from a fitted pliable object Make predictions from a fitted pliable lasso model*

Description

Compute predicted values from a fitted pliable object Make predictions from a fitted pliable lasso model

Usage

```
## S3 method for class 'pliable'
predict(object, x, z, type = c("link", "response",
  "coefficients"), lambda = NULL, verbose = FALSE, ...)
```

Arguments

object	object returned from a call to pliable
x	n by p matrix of predictors
z	n by nz matrix of modifying variables. These may be observed or the predictions from a supervised learning algorithm that predicts z from test features x and possibly other features. See example below
type	Returns either the fitted values with type="link" or "response" or the parameter estimates when type="coefficients" . Type "link" gives the linear predictors for binomial, or cox models; for gaussian models it gives the fitted values. Type "response" gives the fitted probabilities for binomial, and the fitted relative-risk for cox; for gaussian type "response" is equivalent to type "link".
lambda	values of lambda at whcih predictions are desired. If NULL (default), the path of lambda values from the fitted model. are used. If lambda is not NULL, the predictions are made at the closest values to lambda in the lambda path from the fitted model;
verbose	Should information should be printed along the way? Default FALSE.
...	Further arguments (not used) @return predicted values

Examples

```
# Train a pliable lasso model
n = 20; p = 3 ;nz=3
x = matrix(rnorm(n*p), n, p)
z =matrix(rnorm(n*nz),n,nz)
y = x[,1] +x[,1]*z[,3]+ rnorm(n)
  fit = pliable(x,z,y)
 # plot coefficient profiles, indicating z-interactions with a "x" symbol
plot(fit)


# Predict using the fitted model
ntest=500
xtest = matrix(rnorm(ntest*p),ntest,p)
ztest =matrix(rnorm(ntest*nz),ntest,nz)

pred= predict(fit,xtest,ztest)
```

```
 #Example where z is not observed in the test set,  but predicted from a  supervised
  #  learning method

 library(glmnet)
n = 20; p = 3 ;nz=3
x = matrix(rnorm(n*p), n, p)
z =matrix(rnorm(n*nz),n,nz)
y = x[,1] +x[,1]*z[,3]+ rnorm(n)
fit = pliable(x,z,y)
# predict z  from x; here we use glmnet, but any other supervised learning method
# could be used
zfit=glmnet(x,z,family="mgaussian")

# Predict using the fitted model
ntest=500
xtest = matrix(rnorm(ntest*p),ntest,p)
ztest =predict(zfit,xtest,s=zfit$lambda.min)[,,20]

pred= predict(fit,xtest,ztest)
```

Index