

# Package ‘gamlr’

June 8, 2020

**Title** Gamma Lasso Regression

**Version** 1.13-6

**Author** Matt Taddy <mataddy@gmail.com>

**Maintainer** Matt Taddy <mataddy@gmail.com>

**Depends** R (>= 2.15), Matrix, methods, graphics, stats

**Suggests** parallel

## Description

The gamma lasso algorithm provides regularization paths corresponding to a range of non-convex cost functions between L0 and L1 norms. As much as possible, usage for this package is analogous to that for the glmnet package (which does the same thing for penalization between L1 and L2 norms). For details see: Taddy (2017 JCGS), 'One-Step Estimator Paths for Concave Regularization', <arXiv:1308.5623>.

**License** GPL-3

**URL** <http://github.com/TaddyLab/gamlr>

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2020-06-08 21:50:02 UTC

## R topics documented:

AICc . . . . .	2
cv.gamlr . . . . .	3
gamlr . . . . .	5
hockey . . . . .	8

<b>Index</b>	<b>10</b>
--------------	-----------

---

AICc

*Corrected AIC*

---

### Description

Corrected AIC calculation.

### Usage

```
AICc(object, k=2)
```

### Arguments

object	Some model object that you can call logLik on (such as a gamlr or glm fit).
k	The usual AIC complexity penalty. k defaults to 2.

### Details

This works just like usual AIC, but instead calculates the small sample (or high dimensional) corrected version from Hurvich and Tsai

$$AICc = -2 \log LHD + k * df * \frac{n}{n - df - 1}.$$

### Value

A numeric value for every model evaluated.

### Author(s)

Matt Taddy <mataddy@gmail.com>

### References

Hurvich, C. M. and C-L Tsai, 1989. "Regression and Time Series Model Selection in Small Samples", Biometrika 76.

### See Also

gamlr, hockey

**Description**

Cross validation for gamma lasso penalty selection.

**Usage**

```
cv.gamlr(x, y, nfold=5, foldid=NULL, verb=FALSE, cl=NULL, ...)
## S3 method for class 'cv.gamlr'
plot(x, select=TRUE, df=TRUE, ...)
## S3 method for class 'cv.gamlr'
coef(object, select=c("1se","min"), ...)
## S3 method for class 'cv.gamlr'
predict(object, newdata, select=c("1se","min"), ...)
```

**Arguments**

x	Covariates; see gamlr.
y	Response; see gamlr.
nfold	The number of cross validation folds.
foldid	An optional length-n vector of fold memberships for each observation. If specified, this dictates nfold.
verb	Whether to print progress through folds.
cl	possible parallel library cluster. If this is not-NULL, the CV folds are executed in parallel. This copies the data nfold times, so make sure you have the memory space.
...	Arguments to gamlr.
object	A gamlr object.
newdata	New x data for prediction.
select	In prediction and coefficient extraction, select which "best" model to return: select="min" is that with minimum average OOS deviance, and select="1se" is that whose average OOS deviance is no more than 1 standard error away from the minimum. In plot, whether to draw these selections.
df	Whether to add to the plot degrees of freedom along the top axis.

**Details**

Fits a gamlr regression to the full dataset, and then performs nfold cross validation to evaluate out-of-sample (OOS) performance for different penalty weights.

plot.cv.gamlr can be used to plot the results: it shows mean OOS deviance with 1se error bars.

**Value**

<code>gamlr</code>	The full-data fitted <code>gamlr</code> object.
<code>ifold</code>	The number of CV folds.
<code>foldid</code>	The length- <code>n</code> vector of fold memberships.
<code>cvm</code>	Mean OOS deviance by <code>gamlr\$lambda</code>
<code>cvs</code>	The standard errors on <code>cvm</code> .
<code>seg.min</code>	The index of minimum <code>cvm</code> .
<code>seg.1se</code>	The index of 1se <code>cvm</code> (see details).
<code>lambda.min</code>	Penalty at minimum <code>cvm</code> .
<code>lambda.1se</code>	Penalty at 1se <code>cvm</code> .

**Author(s)**

Matt Taddy <mataddy@gmail.com>

**References**

Taddy (2017 JCGS), One-Step Estimator Paths for Concave Regularization, <http://arxiv.org/abs/1308.5623>

**See Also**

`gamlr`, `hockey`

**Examples**

```
n <- 100
p <- 100

xvar <- matrix(ncol=p,nrow=p)
for(i in 1:p) for(j in i:p) xvar[i,j] <- 0.5^{abs(i-j)}
x <- matrix(rnorm(p*n), nrow=n)%*%chol(xvar)
beta <- matrix( (-1)^(1:p)*exp(-(1:p)/10) )
mu = x%*%beta
y <- mu + rnorm(n,sd=sd(as.vector(mu))/2)

## fit with gamma=1 concavity
cvfit <- cv.gamlr(x, y, gamma=1, verb=TRUE)

coef(cvfit)[1:3,] # 1se default
coef(cvfit, select="min")[1:3,] # min OOS deviance

predict(cvfit, x[1:2,], select="min")
predict(cvfit$gamlr, x[1:2,], select=cvfit$seg.min)

par(mfrow=c(1,2))
plot(cvfit)
plot(cvfit$gamlr)
```

**Description**

Adaptive L1 penalized regression estimation.

**Usage**

```
gamlr(x, y,
      family=c("gaussian", "binomial", "poisson"),
      gamma=0, nlambda=100, lambda.start=Inf,
      lambda.min.ratio=0.01, free=NULL, standardize=TRUE,
      obsweight=NULL, varweight=NULL,
      prexx=(p<500),
      tol=1e-7, maxit=1e5, verb=FALSE, ...)

## S3 method for class 'gamlr'
plot(x, against=c("pen", "dev"),
     col=NULL, select=TRUE, df=TRUE, ...)
## S3 method for class 'gamlr'
coef(object, select=NULL, k=2, corrected=TRUE, ...)
## S3 method for class 'gamlr'
predict(object, newdata,
        type = c("link", "response"), ...)
## S3 method for class 'gamlr'
logLik(object, ...)
```

**Arguments**

<code>x</code>	A dense matrix or sparse Matrix of covariates, with <code>ncol(x)</code> variables and <code>nrow(x)==length(y)</code> observations. This should not include the intercept.
<code>y</code>	A vector of response values. There is almost no argument checking, so be careful to match <code>y</code> with the appropriate family
<code>family</code>	Response model type; either "gaussian", "poisson", or "binomial". Note that for "binomial", <code>y</code> is in $[0, 1]$ .
<code>gamma</code>	Penalty concavity tuning parameter; see details. Zero (default) yields the lasso, and higher values correspond to a more concave penalty.
<code>nlambda</code>	Number of regularization path segments.
<code>lambda.start</code>	Initial penalty value. Default of <code>Inf</code> implies the infimum <code>lambda</code> that returns all zero coefficients. This is the largest absolute coefficient gradient at the null model.
<code>lambda.min.ratio</code>	The smallest penalty weight (expected L1 cost) as a ratio of the path start value. Our default is always 0.01; note that this differs from <code>glmnet</code> whose default depends upon the dimension of <code>x</code> .

free	Free variables: indices of the columns of $x$ which will be unpenalized.
standardize	Whether to standardize the coefficients to have standard deviation of one. This is equivalent to multiplying the L1 penalty by each coefficient standard deviation.
obsweight	For family="gaussian" only, weights on each observation in the weighted least squares objective. For other response families, obsweights are overwritten by IRLS weights. Defaults to <code>rep(1, n)</code> .
varweight	Multipliers on the penalty associated with each covariate coefficient. Must be non-negative. These are further multiplied by $sd(x_j)$ if <code>standardize=TRUE</code> . Defaults to <code>rep(1, p)</code> .
prexx	Only possible for family="gaussian": whether to use pre-calculated weighted variable covariances in gradient calculations. This leads to massive speed-ups for big-n datasets, but can be slow for $p > n$ datasets. See note.
tol	Optimization convergence tolerance relative to the null model deviance for each inner coordinate-descent loop. This is measured against the maximum coordinate change times deviance curvature after full parameter-set update.
maxit	Max iterations for a single segment coordinate descent routine.
verb	Whether to print some output for each path segment.
object	A <code>gamlr</code> object.
against	Whether to plot paths against log penalty or deviance.
select	In <code>coef</code> (and <code>predict</code> , which calls <code>coef</code> ), the index of path segments for which you want coefficients or prediction (e.g., do <code>select=which.min(BIC(object))</code> for BIC selection). If null, the segments are selected via our AICc function with <code>k</code> as specified (see also <code>corrected</code> ). If <code>select=0</code> all segments are returned. In <code>plot</code> , <code>select</code> is just a flag for whether to add lines marking AICc and BIC selected models.
k	If <code>select=NULL</code> in <code>coef</code> or <code>predict</code> , the AICc complexity penalty. <code>k</code> defaults to the usual 2.
corrected	A flag that swaps corrected (for high dimensional bias) AICc in for the standard AIC. You almost always want <code>corrected=TRUE</code> , unless you want to apply the BIC in which case use <code>k=log(n)</code> , <code>corrected=FALSE</code> .
newdata	New $x$ data for prediction.
type	Either "link" for the linear equation, or "response" for predictions transformed to the same domain as $y$ .
col	A single plot color, or vector of length <code>ncol(x)</code> colors for each coefficient regularization path. NULL uses the <code>matplot</code> default <code>1:6</code> .
df	Whether to add to the plot degrees of freedom along the top axis.
...	Extra arguments to each method. Most importantly, from <code>predict.gamlr</code> these are arguments to <code>coef.gamlr</code> .

## Details

Finds posterior modes along a regularization path of *adapted L1 penalties* via coordinate descent.

Each path segment  $t$  minimizes the objective  $-(\phi/n)\log\text{LHD}(\beta_1\dots\beta_p) + \sum \omega_j \lambda |\beta_j|$ , where  $\phi$  is the exponential family dispersion parameter ( $\sigma^2$  for family="gaussian", one otherwise). Weights  $\omega_j$  are set as  $1/(1 + \gamma|b_j^{t-1}|)$  where  $b_j^{t-1}$  is our estimate of  $\beta_j$  for the previous path segment (or zero if  $t = 0$ ). This adaptation is what makes the penalization 'concave'; see Taddy (2013) for details.

`plot.gamlr` can be used to graph the results: it shows the regularization paths for penalized  $\beta$ , with degrees of freedom along the top axis and minimum AICc selection marked.

`logLik.gamlr` returns log likelihood along the regularization path. It is based on the deviance, and is correct only up to static constants; e.g., for a Poisson it is off by  $\sum_i y_i (\log y_i - 1)$  (the saturated log likelihood) and for a Gaussian it is off by likelihood constants  $(n/2)(1 + \log 2\pi)$ .

### Value

lambda	The path of fitted <i>prior expected</i> L1 penalties.
nobs	The number of observations.
alpha	Intercepts.
beta	Regression coefficients.
df	Approximate degrees of freedom.
deviance	Fitted deviance: $(-2/\phi)(\log\text{LHD.fitted} - \log\text{LHD.saturated})$ .
iter	Number of optimization iterations by segment, broken into coordinate descent cycles and IRLS re-weightings for family!="gaussian".
family	The exponential family model.

### Note

Under `prexx=TRUE` (requires family="gaussian"), weighted covariances  $(VX)'X$  and  $(VX)'y$ , weighted column sums of  $VX$ , and column means  $\bar{x}$  will be pre-calculated. Here  $V$  is the diagonal matrix of least squares weights (obsweights, so  $V$  defaults to  $I$ ). It is not necessary (they will be built by `gamlr` otherwise), but you have the option to pre-calculate these sufficient statistics yourself as arguments `vxx` (matrix or `dspMatrix`), `vxy`, `vxsum`, and `xbar` (all vectors) respectively. Search `PREXX` in `gamlr.R` to see the steps involved, and notice that there is very little argument checking – do at your own risk. Note that `xbar` is an *unweighted* calculation, even if  $V \neq I$ . For really Big Data you can then run with `x=NULL` (e.g., if these statistics were calculated on distributed machines and full design is unavailable). *Beware*: in this `x=NULL` case our deviance (and `df`, if `gamma>0`) calculations are incorrect and selection rules will always return the smallest-lambda model.

### Author(s)

Matt Taddy <mataddy@gmail.com>

### References

Taddy (2017 JCGS), One-Step Estimator Paths for Concave Regularization, <http://arxiv.org/abs/1308.5623>

### See Also

`cv.gamlr`, AICc, `hockey`

## Examples

```
### a low-D test (highly multi-collinear)

n <- 1000
p <- 3
xvar <- matrix(0.9, nrow=p, ncol=p)
diag(xvar) <- 1
x <- matrix(rnorm(p*n), nrow=n)%*%chol(xvar)
y <- 4 + 3*x[,1] + -1*x[,2] + rnorm(n)

## run models to extra small lambda 1e-3xlambda.start
fitlasso <- gamlr(x, y, gamma=0, lambda.min.ratio=1e-3) # lasso
fitgl <- gamlr(x, y, gamma=2, lambda.min.ratio=1e-3) # small gamma
fitglbv <- gamlr(x, y, gamma=10, lambda.min.ratio=1e-3) # big gamma

par(mfrow=c(1,3))
ylim = range(c(fitglbv$beta@x))
plot(fitlasso, ylim=ylim, col="navy")
plot(fitgl, ylim=ylim, col="maroon")
plot(fitglbv, ylim=ylim, col="darkorange")
```

---

hockey

*NHL hockey data*

---

## Description

Every NHL goal from fall 2002 through the 2014 cup finals.

## Details

The data comprise of information about play configuration and the players on ice (including goalies) for every goal from 2002-03 to 2012-14 NHL seasons. Collected using A. C. Thomas's `nlhscrapr` package. See the Chicago hockey analytics project at [github.com/mataddy/hockey](https://github.com/mataddy/hockey).

## Value

goal	Info about each goal scored, including homegoal – an indicator for the home team scoring.
player	Sparse Matrix with entries for who was on the ice for each goal: +1 for a home team player, -1 for an away team player, zero otherwise.
team	Sparse Matrix with indicators for each team*season interaction: +1 for home team, -1 for away team.
config	Special teams info. For example, S5v4 is a 5 on 4 powerplay, +1 if it is for the home-team and -1 for the away team.



**Author(s)**

Matt Taddy, <mataddy@gmail.com>

**References**

Gramacy, Jensen, and Taddy (2013): "Estimating Player Contribution in Hockey with Regularized Logistic Regression", the Journal of Quantitative Analysis in Sport.

Gramacy, Taddy, and Tian (2015): "Hockey Player Performance via Regularized Logistic Regression", the Handbook of statistical methods for design and analysis in sports.

**See Also**

gamlr

**Examples**

```
## design
data(hockey)
x <- cbind(config,team,player)
y <- goal$homegoal

## fit the plus-minus regression model
## (non-player effects are unpenalized)

fit <- gamlr(x, y,
  lambda.min.ratio=0.05, nlambda=40, ## just so it runs in under 5 sec
  free=1:(ncol(config)+ncol(team)),
  standardize=FALSE, family="binomial")
plot(fit)

## look at estimated player [career] effects
B <- coef(fit)[colnames(player),]
sum(B!=0) # number of measurable effects (AICc selection)
B[order(-B)[1:10]] # 10 biggest

## convert to 2013-2014 season partial plus-minus
now <- goal$season=="20132014"
pm <- colSums(player[now,names(B)]*c(-1,1)[y[now]+1]) # traditional plus minus
ng <- colSums(abs(player[now,names(B)])) # total number of goals
# The individual effect on probability that a
# given goal is for vs against that player's team
p <- 1/(1+exp(-B))
# multiply ng*p - ng*(1-p) to get expected plus-minus
ppm <- ng*(2*p-1)

# organize the data together and print top 20
effect <- data.frame(b=round(B,3),ppm=round(ppm,3),pm=pm)
effect <- effect[order(-effect$ppm),]
print(effect[1:20,])
```

# Index

AICc, [2](#)

`coef.cv.gamlr (cv.gamlr)`, [3](#)

`coef.gamlr (gamlr)`, [5](#)

`config (hockey)`, [8](#)

`cv.gamlr`, [3](#)

`gamlr`, [5](#)

`goal (hockey)`, [8](#)

`hockey`, [8](#)

`logLik.gamlr (gamlr)`, [5](#)

`player (hockey)`, [8](#)

`plot.cv.gamlr (cv.gamlr)`, [3](#)

`plot.gamlr (gamlr)`, [5](#)

`predict.cv.gamlr (cv.gamlr)`, [3](#)

`predict.gamlr (gamlr)`, [5](#)

`team (hockey)`, [8](#)