Using glmnetr

Walter K. Kremers, Mayo Clinic, Rochester MN

28 May 2023

The Package

The *glmnetr* packages fits ralxed lasso, artificial neural network (NN), gradient boosting machine (GBM), resursive partitioning and stepwise regression models, all with hyperparameters informed by cross validation. It fits all these model as extensions of linear, logistic and Cox regression models. The package can fit all these models in a single call, and performs nested cross validation allowing the user to evaluate and compare the performances of these different models. The package fits these models using other r packages including *glmnet*, *survival*, *xgboost*, *rpart* and *torch*. For the relaxed lasso models *glmnetr* uses R *stat* and *survival* to obtain stable model fits, and obtain these often more quickly. This too might be achieved using the 'path=TRUE' option in *glmnet*.

While the package fits nested cross validation for the lasso and other models, it does not fit the general elastic net model. If you are fitting not a relaxed lasso model but an elastic-net model, then the R-packages *nestedcv* (https://cran.r-project.org/package=nestedcv), 'glmnetSE' (https://cran.r-project.org/package=glmnetSE) or others may provide greater functionality when performing a nested CV.

As with the *glmnet* package, this package passes most relevant information to the output object which can be evaluated using plot, summary() and predict() functions. The *glmnetr* package has some features and functionality that we find useful, but omits some of the functionality of *glmnet* as well. Use of the *glmnetr* package has many similarites to the *glmnet* package and it is recommended that the user of *glmnetr* first become familiar with the *glmnet* package (https://cran.r-project.org/package=glmnet), with the "An Introduction to glmnet" and "The Relaxed Lasso" being especially helpful in this regard.

Data requirements

The basic data elements for input to the *glmnetr* analysis programs are similar to those of *glmnet* and include 1) a matrix of predictors and 2) an outcome variable or variables in vector form. For the estimation of the "fully" relaxed models (where gamma=0) the package is set up to fit the "gaussian" and "binomial" models using the *stats* glm() function and Cox survival models using the the coxph() function of the *survival* pacakge. When fitting these extensions to the Cox model the outcome model variable is interpreted as the "time" variable in the Cox model, and one must also specify 3) a variable for event, again in vector form, and optionally 4) a variable for start time, also in vector form. Row i of the predictor matrix and element i of the outcome vector(s) are to include the data for the same sampling unit.

An example dataset

To demonstrate usage of *glmnetr* we first generate a data set for analysis, run an analysis and evaluate using the plot(), summary() and predict() functions.

The code

```r
# Simulate data for use in an example relaxed lasso fit of survival data
# first, optionally, assign a seed for random number generation to get replicable results
set.seed(829662260)
simdata=glmnetr.simdata(nrows=1000, ncols=100, beta=NULL, intr=c(1,0,1,1))
```

generates simulated data for analysis. We extract data in the format required for input to the *glmnetr* programs as in

```r
# Extract simulated survival data
xs = simdata$xs         # matrix of predictors
y_ = simdata$yt         # vector of survival times
event = simdata$event   # indicator of event vs. censoring
```

Inspecting the predictor matrix we see

```r
# Check the sample size and number of predictors
print(dim(xs))
```

```
## [1] 1000  100
```

```r
# Check the rank of the design matrix, i.e. the degrees of freedom in the predictors
rankMatrix(xs)[[1]]
```

```
## [1] 94
```

```r
# Inspect the first few rows and some select columns
print(xs[1:10,c(1:12,18:20)])
```

```
##       X1 X2 X3 X4 X5 X6 X7 X8 X9 X10 X11 X12        X18        X19        X20
##  [1,]  1  1  0  0  0  0  0  1  0   0   0   0 -0.1945924  0.9478287  0.4842656
##  [2,]  1  1  0  0  0  0  0  0  0   0   1   0  0.3569468  0.3189536  0.6691443
##  [3,]  1  0  0  0  1  0  0  1  0   0   0   1  1.0863620 -1.6320687  0.4900879
##  [4,]  1  0  0  1  0  0  0  1  0   0   0   0  0.3790386 -0.5276492 -0.7512417
##  [5,]  1  0  0  0  1  0  1  0  0   0   0   0 -0.1144306  1.2557999 -1.1179235
##  [6,]  1  1  0  0  0  0  0  0  0   1   0   0 -1.2715046  0.2836655  1.1918969
##  [7,]  1  0  0  0  1  0  0  1  0   0   0   0  1.7616709 -1.3564057 -0.1199141
##  [8,]  1  0  1  0  0  0  0  0  1   0   0   0  0.1972549  0.3076172 -2.1272796
##  [9,]  1  0  0  1  0  1  0  0  0   0   0   0 -0.5846613  0.9945884 -1.2646171
## [10,]  1  0  0  0  1  0  1  0  0   0   0   0  1.1677768  0.2320757  1.3743778
```

Cross validation (CV) informed relaxed lasso fit

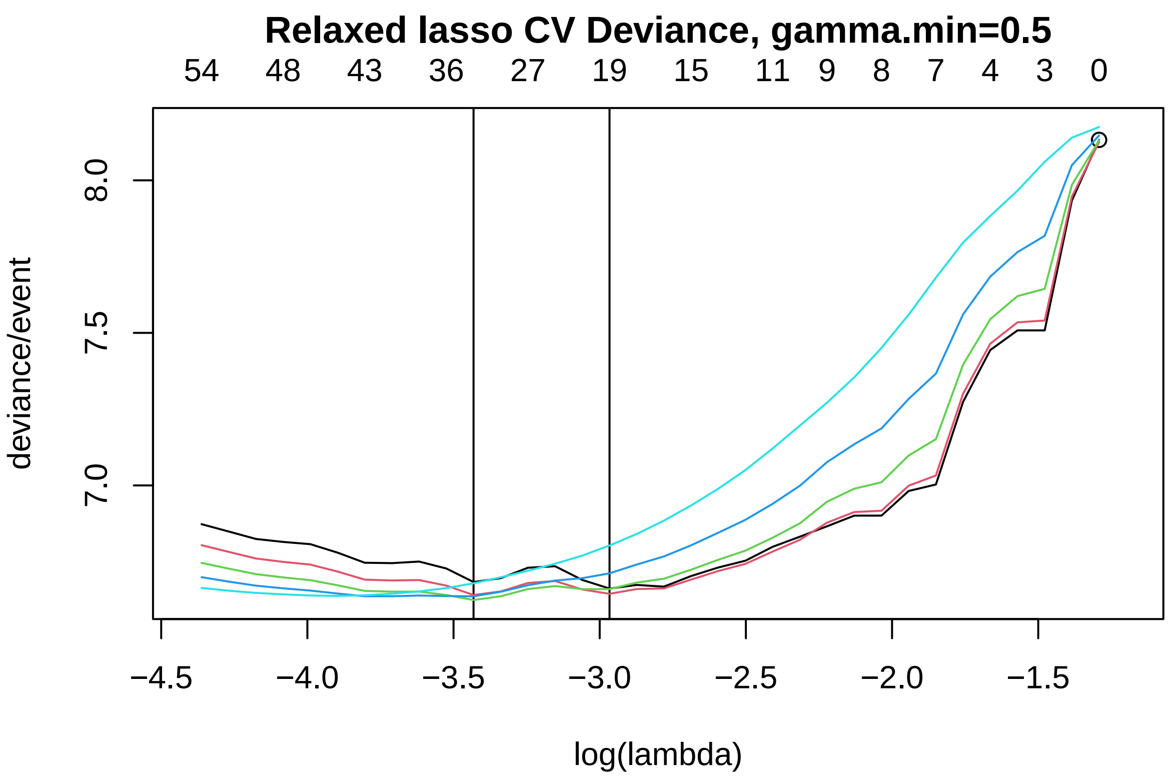
To fit a cross-validated "tuned" relaxed lasso model can use the cv.glmnetr() function. In this example we fit a Cox regression model extension but linear and logistic regression model extension fits can be carried out analogously.

```r
# Fit a relaxed lasso model informed by cross validation
cv.cox.fit = suppressWarnings( cv.glmnetr(xs, NULL, y_, event, family="cox", track=0) )
```

Note, in the derivation of the relaxed lasso model fits, individual coefficients may be unstable even when the model may be stable which elicits warning messages. Thus we "wrapped" the call to cv.glmnetr() within the suppressWarnings() function to avoid excessive warning messages in this vignette. The first input term in the call to cv.glmnetr(), xs, is the design matrix for predictors. The second input term, here NULL, is for the start time in case (start, stop) time data setup is used in a Cox survival model. The third term is the outcome variable for the linear regression or logistic regression model and the time of event or censoring in case of the Cox model, and finally the forth term is the event indicator variable for the Cox model taking the value 1 in case of an event or 0 in case of censoring at time y_. The forth term would be NULL for either linear or logistic regression. Currently the options for family are "guassian" for linear regression, "binomial" for logistic regression (both using the *stats* glm() function) and "cox" for the Cox proportional hazards regression model using the coxph() function of the R *survival* package. If one sets track=1 the program will update progress in the R console, else for track=0 it will not. | Before numerically summarizing the model fit, or inspecting the coefficient estimates, we plot the average deviances using the plot() function.

```
# Plot cross validation average deviances for a relaxed lasso model
plot(cv.cox.fit)
```

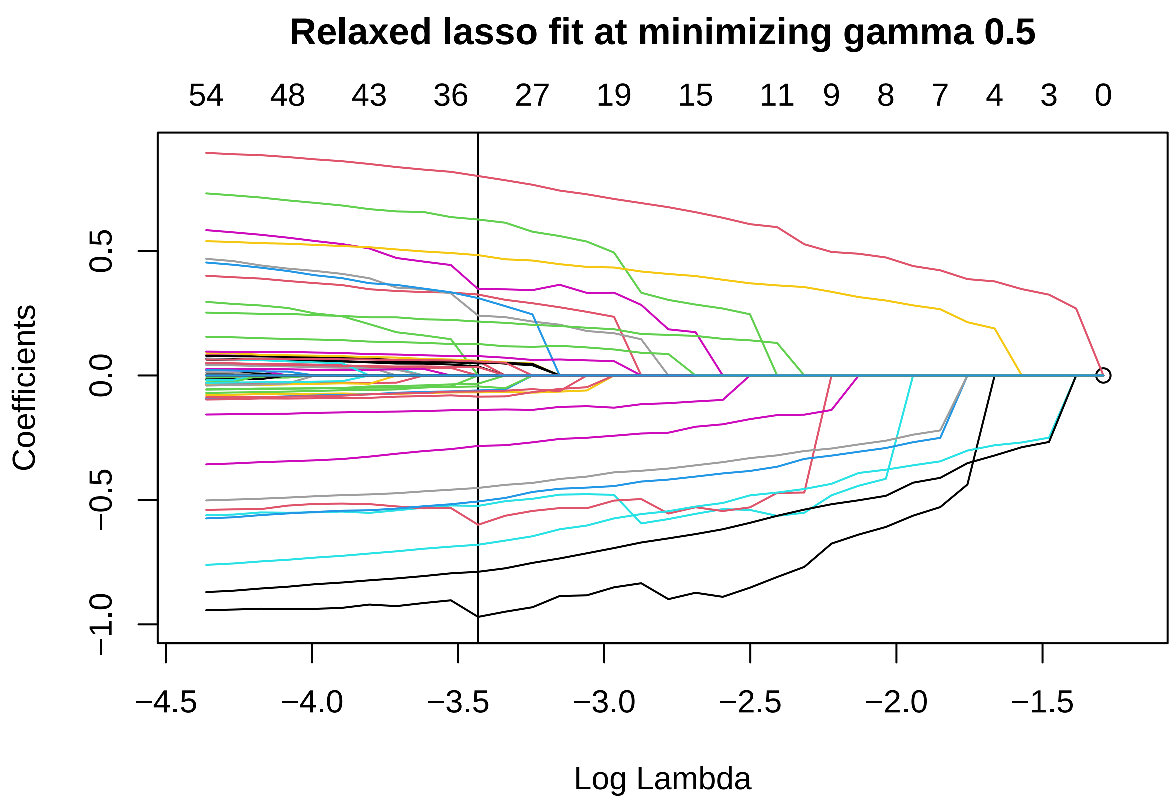
```
##  min CV average deviance (max log likelihood) for
##    relaxed at log(lambda)  = -3.431, gamma.min = 0.5, df = 34
##    fully relaxed at log(lambda)   = -2.966, df = 19
##    fully penalized at log(lambda) = -3.896, df = 44
```



In that to maximize the log-likelihoods is to minimize deviance we inspect these curves for a minimum. The minimizing lambda is indicated by the left most vertical line, here about log(lambda) = -3.43. The minimizing gamma is 0.5 and described in the title. Whereas there is no legend here for gamma, when non-zero coefficients start to enter the model as when the penaly is large, here shown to the right, deviances will tend to be smaller for gamma = 0, greater for gamma = 1 and in between for other gammas values. From this figure we also see that at lambda=0.5 the deviances are very similar for the best models amongst those with gamma ranging from 0.5 to 1. More relevant we see that the fully relaxed lasso (gamma=0) and indicated by the right most vertical line, achieves a "nearly" minimal deviance at about -2.97.

```
# Plot coefficients informed by a cross validation
plot(cv.cox.fit, coefs=TRUE)
```

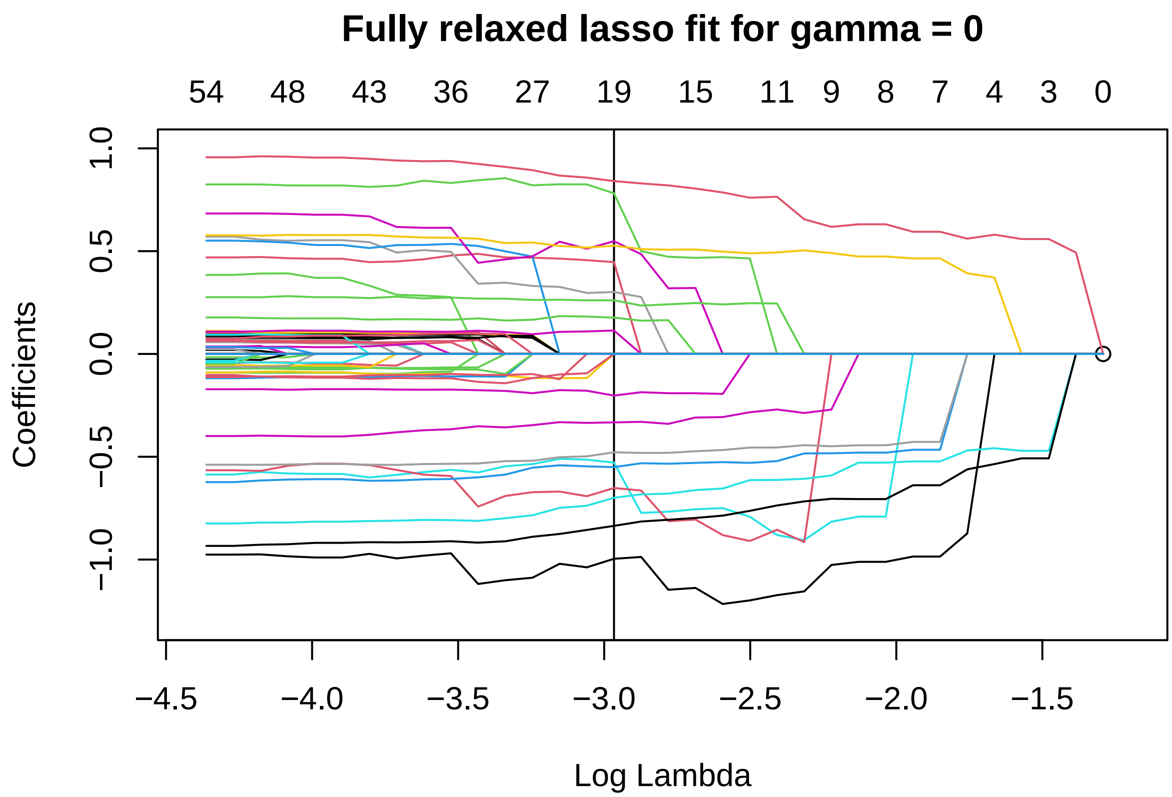
```
##  min CV average deviance (max log likelihood)
##   at log(lambda.min) = -3.432, gamma.min = 0.5, df = 34
```



In this plot of coefficients we use the same orientation for lambda as in the plot for deviances with larger values of the lambda penalty to the right and corresponding to fewer non-zero coefficients. The displayed coefficients are for the minimizing gamma=0.5 as noted in the tile, and the minimizing lambda indicated by the vertical line. Now, since the fully relaxed lasso model had a deviance almost that of the relaxed lasso model we also plot the coefficients using the option gam=0.

```
# Plot fully relaxed coefficients informed by a cross validation
plot(cv.cox.fit, coefs=TRUE, gam=0)
```

```
##  Fully relaxed min CV average deviance (max log likelihood)
##   at log(lambda.min) = -2.966, df = 19
```



This plot shows how the coefficients change for the un-penalized (fully relaxed) model with gamma=0 as lambda decreases. In particular we see how new terms enter into the model as the lamba penalty decreases, the coefficients in general become slightly larger as lambda decreases while some terms decrease as lambda decreases. This is not unexpected as omitted terms from the Cox model tend to bias coefficients toward 0 more than increase the standard error. We also see the number of model non-zero coefficients, 19, to be substantially less than the 34 from the relaxed lasso fit and the 44 from the fully penalized lasso fit.
| The summary() function describes the relaxed lasso fit informed by CV.

```
# Summarize relaxed lasso model fit informed by cross validation
summary(cv.cox.fit)
```

```
##
##   The relaxed minimum is obtained for lambda = 0.03233462 , index = 24 and gamma = 0.5
##   with df (number of non-zero terms) = 34, average deviance = 6.624441 and beta =
##             X2            X3            X5            X8            X9
##   3.252630e-01  6.267567e-01 -5.240280e-01  2.406727e-01 -9.699629e-01
##            X10           X12           X13           X14           X15
## -5.998392e-01  3.109241e-01 -1.807644e-14  3.472129e-01 -3.366661e-16
##            X18           X19           X20           X21           X22
##   8.014231e-01  2.166607e-01 -5.060636e-01 -6.799914e-01 -2.830400e-01
##            X23           X24           X25           X26           X30
##   4.835785e-01 -4.518058e-01 -7.887317e-01  5.770815e-02 -1.381467e-01
##            X33           X39           X44           X50           X55
##   5.594976e-02 -6.780871e-02 -6.008669e-02 -8.502705e-02  5.934026e-02
##            X57           X67           X70           X73           X74
##   3.647977e-02 -4.396762e-02  7.779270e-02  4.834927e-02 -6.421541e-02
##            X75           X90           X98           X99
```

```
## -3.293569e-02  5.618672e-02  3.388149e-02  1.264423e-01
##
##   The fully relaxed (gamma=0) minimum is obtained for lambda = 0.05148586 and index = 19
##   with df (number of non-zero terms) = 18, average deviance = 6.661994 and beta =
##         X2         X3         X5         X8         X9        X10        X14
##  0.4464710  0.7814314 -0.5284294  0.3012989 -0.9963434 -0.6520494  0.5484414
##        X18        X19        X20        X21        X22        X23        X24
##  0.8407381  0.2609653 -0.5505213 -0.6993674 -0.3328967  0.5266075 -0.4782524
##        X25        X30        X70        X99
## -0.8360974 -0.2023634  0.1138980  0.1769053
##
##   The UNrelaxed (gamma=1) minimum is obtained for lambda = 0.02030709 and index = 29
##   with df (number of non-zero terms) = 44, average deviance = 6.637919
##
##
##   Order coefficients entered into the lasso model (1st to last):
##  [1] "X18" "X21" "X25" "X23" "X9"  "X20" "X24" "X5"  "X22" "X10" "X19" "X3"
## [13] "X30" "X14" "X99" "X8"  "X2"  "X70" "X39" "X50" "X74" "X12" "X33" "X55"
## [25] "X73" "X44" "X67" "X90" "X26" "X57" "X75" "X98" "X11" "X27" "X58" "X78"
## [37] "X42" "X45" "X48" "X56" "X79" "X85" "X93" "X35"
```

In the summary output we first see the relaxed lasso model fit based upon the (lambda, gamma) pair which minimizes the cross validated average deviance. Next is the model fit based upon the lambda that minimizes the cross validated average deviance along the path where gamma=0, that is among the fully relaxed lasso models. After that is information on the fully penalized lasso fit, but without the actual coefficient estimates. These estimates can be printed using the option *printg1=TRUE*, but are suppressed by default for space. Finally, the order that coefficients enter the lasso model as the penalty is decreased is provided, which gives some indication of relative model importance of the coefficients. Because, though, the differences in successive lambda values used in the numerical algorithms may allow multiple new terms to enter into the model between successive numerical steps, the ordering in this list may not be strict. If the user would want they could read lambda from output$lambda, set up a new lambda with finer steps and rerun the model. Our experience though is that this does not generally lead to a meaningfully different model and so is not done by default or as option. | One can as well use the predict() function to get the coefficients for the lasso model, or the xs_new*beta for a new design matrix xs_new. In contrast to the summary() function which simply displays coefficients, the predict() function provides an outpout object in vector form (or a list with two vectors) and so can more easily be used for further calculations. By default the summary() function will use the (lambda, gamma) pair that minimizes the average CV deviances. One can also specify lam=NULL and gam=1 to use the fully penalized lasso best fit, that use the solution that minimizes the CV deviance with respect to lambda while holding gamma=1, or gam=0 to use the fully relaxed lasso best fit, that is minimizes while holding gamma=0. One can also numerically specify both lam for lambda and gam for gamma. Within the package lambda and gamma usually denote vectors for the search algorithm and so other names are used uere.

```
# Get coefficients
beta = predict(cv.cox.fit)
```

```
##
##  (lambda, gamma) pair minimizing CV average deviance is used
```

```
# Print out the non-zero coefficients
beta$beta
```

```
##            X2            X3            X5            X8            X9
```

```
##  3.252630e-01  6.267567e-01 -5.240280e-01  2.406727e-01 -9.699629e-01
##           X10           X12           X13           X14           X15
## -5.998392e-01  3.109241e-01 -1.807644e-14  3.472129e-01 -3.366661e-16
##           X18           X19           X20           X21           X22
##  8.014231e-01  2.166607e-01 -5.060636e-01 -6.799914e-01 -2.830400e-01
##           X23           X24           X25           X26           X30
##  4.835785e-01 -4.518058e-01 -7.887317e-01  5.770815e-02 -1.381467e-01
##           X33           X39           X44           X50           X55
##  5.594976e-02 -6.780871e-02 -6.008669e-02 -8.502705e-02  5.934026e-02
##           X57           X67           X70           X73           X74
##  3.647977e-02 -4.396762e-02  7.779270e-02  4.834927e-02 -6.421541e-02
##           X75           X90           X98           X99
## -3.293569e-02  5.618672e-02  3.388149e-02  1.264423e-01
```

```
# Print out all coefficients
beta$beta_
```

```
##            X1            X2            X3            X4            X5
##  0.000000e+00  3.252630e-01  6.267567e-01  0.000000e+00 -5.240280e-01
##            X6            X7            X8            X9           X10
##  0.000000e+00  0.000000e+00  2.406727e-01 -9.699629e-01 -5.998392e-01
##           X11           X12           X13           X14           X15
##  0.000000e+00  3.109241e-01 -1.807644e-14  3.472129e-01 -3.366661e-16
##           X16           X17           X18           X19           X20
##  0.000000e+00  0.000000e+00  8.014231e-01  2.166607e-01 -5.060636e-01
##           X21           X22           X23           X24           X25
## -6.799914e-01 -2.830400e-01  4.835785e-01 -4.518058e-01 -7.887317e-01
##           X26           X27           X28           X29           X30
##  5.770815e-02  0.000000e+00  0.000000e+00  0.000000e+00 -1.381467e-01
##           X31           X32           X33           X34           X35
##  0.000000e+00  0.000000e+00  5.594976e-02  0.000000e+00  0.000000e+00
##           X36           X37           X38           X39           X40
##  0.000000e+00  0.000000e+00  0.000000e+00 -6.780871e-02  0.000000e+00
##           X41           X42           X43           X44           X45
##  0.000000e+00  0.000000e+00  0.000000e+00 -6.008669e-02  0.000000e+00
##           X46           X47           X48           X49           X50
##  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00 -8.502705e-02
##           X51           X52           X53           X54           X55
##  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  5.934026e-02
##           X56           X57           X58           X59           X60
##  0.000000e+00  3.647977e-02  0.000000e+00  0.000000e+00  0.000000e+00
##           X61           X62           X63           X64           X65
##  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
##           X66           X67           X68           X69           X70
##  0.000000e+00 -4.396762e-02  0.000000e+00  0.000000e+00  7.779270e-02
##           X71           X72           X73           X74           X75
##  0.000000e+00  0.000000e+00  4.834927e-02 -6.421541e-02 -3.293569e-02
##           X76           X77           X78           X79           X80
##  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
##           X81           X82           X83           X84           X85
##  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
##           X86           X87           X88           X89           X90
##  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  5.618672e-02
##           X91           X92           X93           X94           X95
```

```
##  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
##           X96           X97           X98           X99          X100
##  0.000000e+00  0.000000e+00  3.388149e-02  1.264423e-01  0.000000e+00
```

```
# Get the predicteds (linear predictors) for the original data set
predicteds = predict(cv.cox.fit, xs)
```

```
##
##  (lambda, gamma) pair minimizing CV average deviance is used
```

```
# Print out the first few predicteds
predicteds[1:20]
```

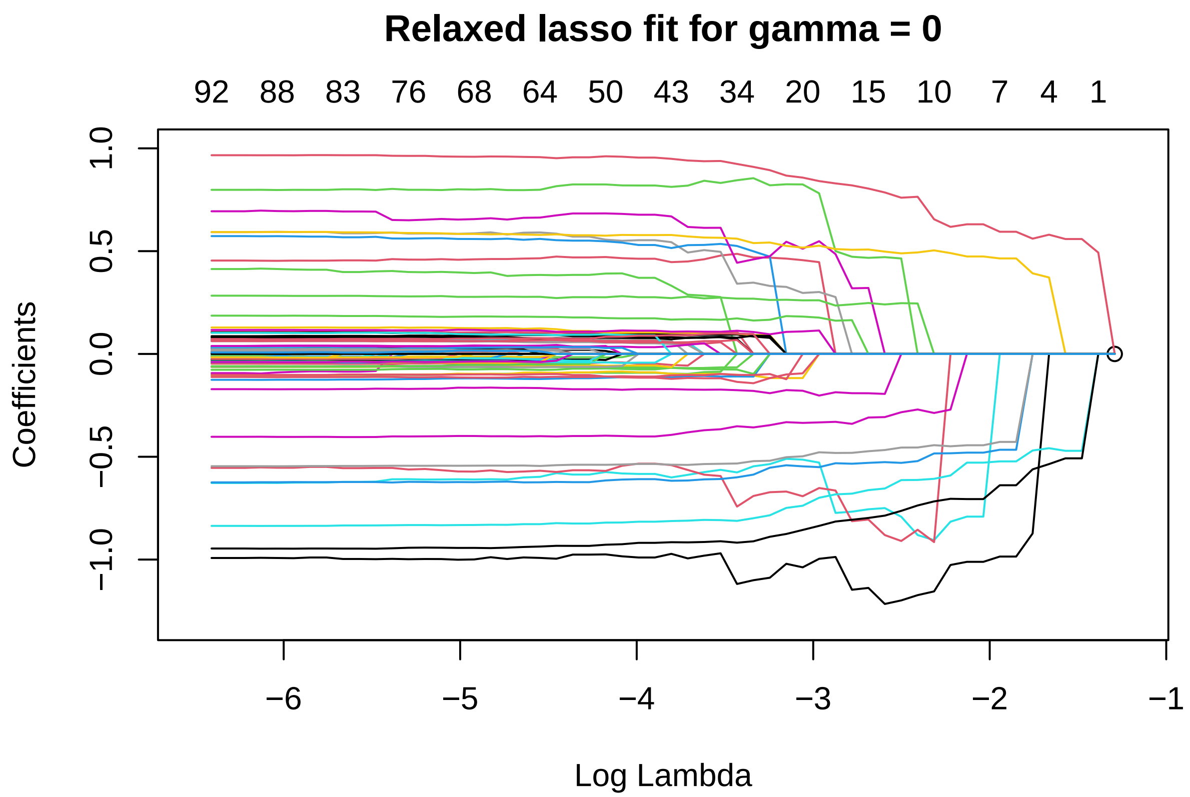
```
##  [1]  1.7917841  0.4225264 -2.5808608 -0.5821384  2.6144458 -0.7552934
##  [7]  0.5794184  0.2546404  2.8613887  1.7483330  2.8752599 -2.4558347
## [13] -3.2428677  0.1303931 -0.7575231  2.2028229 -2.4639038 -0.9912309
## [19]  2.1727112  0.2963720
```

Model fit without cross validation

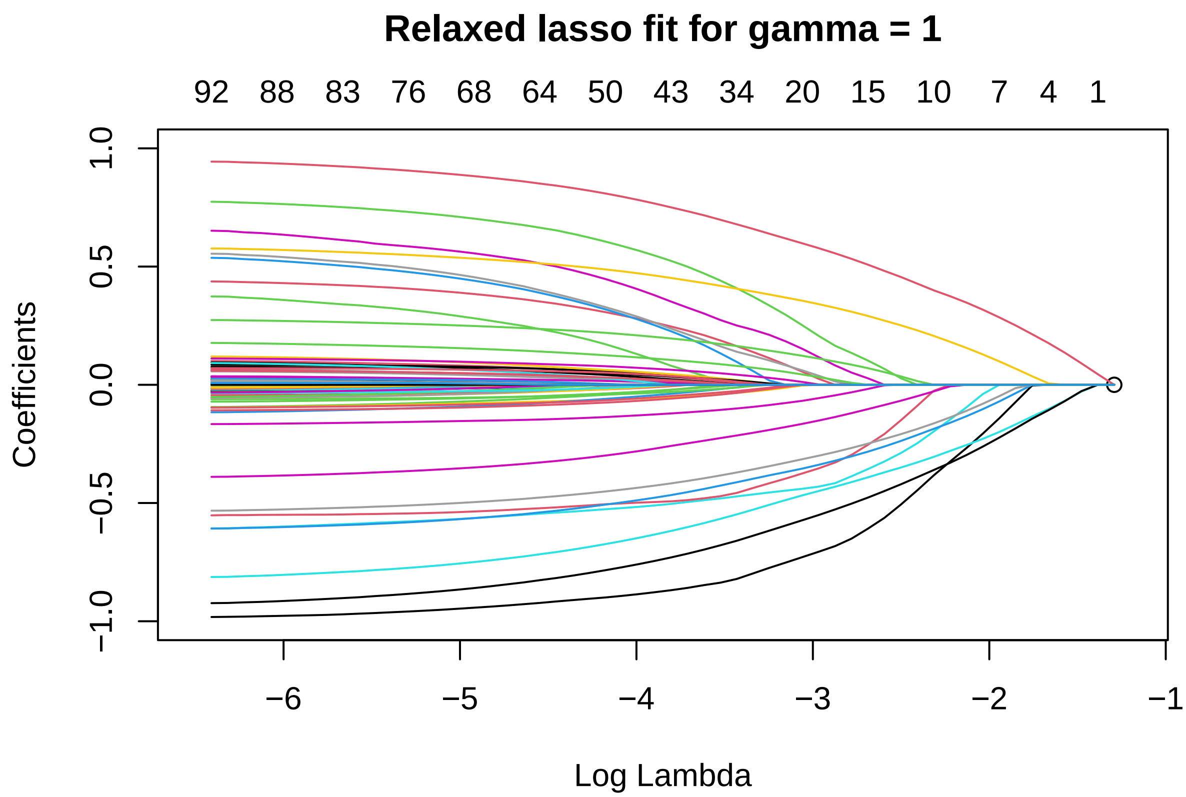
We can as well fit a relaxed lasso model without doing a CV. For this case one can still plot the coefficients but when the minimizing lambda and gamma are not informed by CV one is to specify which gamma should be used for the plots. By default gamma=1, i.e. for the fully penalized lasso model, is used for the plots. One can plot the coefficient estimates for different gamma values, but these will usually be more meaningful when informed by the CV "tuned" hyperparameters values for lambda and gamma. One can also use the predict() function, again to output either coefficients or predicteds, i.e. xs_new*beta for a new design matrix xs_new. Such predicteds are often, for example in coxph(), included in the analysis output object under the name linear.predictors.

```
# Fit a model without cross validation
cox.fit = suppressWarnings( glmnetr(xs, NULL, y_, event, family="cox") )
```

```
# Plot coefficients of the fully relaxed lasso model
plot(cox.fit, gam=0)
```




```
# Plot coefficients of the fully penalized lasso model
plot(cox.fit, gam=1)
```




```
# Get an arbitrary set of coefficients for this example
lam = cox.fit$lambda[min(20,length(cox.fit$lambda))]
predict(cox.fit,lam=lam,gam=1)$beta
```

```
##            X2            X3            X5            X8            X9
##  0.0553432229  0.2516995693 -0.4398532702  0.0608541953 -0.7288383184
##           X10           X14           X18           X19           X20
## -0.3748345705  0.1520232488  0.5983236885  0.1226268322 -0.3543314621
##           X21           X22           X23           X24           X25
## -0.4676224238 -0.1652175787  0.3555883664 -0.3146253292 -0.5728212746
##           X30           X39           X50           X70           X99
## -0.0678040775 -0.0029179790 -0.0001756731  0.0119473856  0.0436750282
```

Nested cross validation fit

Because the values for lambda and gamma informed by CV are specifically chosen to give a best fit, model fit statistics for the CV derived model will be biased. To address this one can perform a CV on the CV derived estimates, that is a nested cross validation as argued for in SRDM (Simon R, Radmacher MD, Dobbin K, McShane LM. Pitfalls in the Use of DNA Microarray Data for Diagnostic and Prognostic Classification. J Natl Cancer Inst (2003) 95 (1): 14-18. https://academic.oup.com/jnci/article/95/1/14/2520188). This is done here by the nested.glmnetr() function. In this example we fit a Cox regression model extension but linear and logistic regression model extension fits can be carried out analogously.

```r
# A nested cross validation to evaluate a cross validation informed lasso model fit
# nested.cox.fit = nested.glmnetr(xs,NULL,y_,event,family="cox",track=1)
nested.cox.fit = suppressWarnings(nested.glmnetr(xs,NULL,y_,event,family="cox",
      doann=1, dorpart=0, ensemble=c(1,0,0,0, 0,1,0,1), folds_n=10, track=0))
summary(nested.cox.fit)
```

```
##
##  Sample information including number of records, events, number of columns in
##  design (predictor, X) matrix, and df (rank) of design matrix:
##          family                n         nevents      xs.columns           xs.df
##           "cox"           "1000"           "587"           "100"            "94"
## null.dev/events
##        "12.614"
##
##  Tuning parameters for models :
## folds_n
##    "10"
##
##  Tuning parameters for  l/lasso update  ANN model :
##   n folds     epochs length Z1 length Z2      actv     drpot      mylr        wd
##    10.000    200.000    18.000    10.000     1.000     0.000     0.001     0.000
##        l1     lscale     scale
##     0.000      5.000     1.000
##
##
##  Nested Cross Validation averages for LASSO (1se and min), Relaxed LASSO, and gamma=0 LASSO :
##
##       deviance per event :
##     1se     min    1seR    minR 1seR.G0 minR.G0   ridge
##   6.641   6.539   6.641   6.562   6.689   6.615   6.789
##
##       deviance per event (linerly calibrated) :
##     1se     min    1seR    minR 1seR.G0 minR.G0   ridge
##   6.500   6.475   6.501   6.491   6.604   6.522   6.600
##
##       number of nonzero model terms :
##     1se     min    1seR    minR 1seR.G0 minR.G0
##    22.5    45.0    19.2    33.4    11.0    17.1
##
##       linear calibration coefficient :
##     1se     min    1seR    minR 1seR.G0 minR.G0   ridge
##   1.398   1.107   1.361   1.078   0.998   0.957   1.584
##
##       agreement (concordance) :
##     1se     min    1seR    minR 1seR.G0 minR.G0   ridge
##   0.849   0.848   0.850   0.847   0.842   0.845   0.836
##
##  Naive agreement for cross validation informed LASSO :
##     1se     min    1seR    minR 1seR.G0 minR.G0   ridge
##   0.855   0.858   0.855   0.858   0.855   0.856   0.859
##
##  Number of non-zero terms in cross validation informed LASSO :
##     1se     min    1seR    minR 1seR.G0 minR.G0   ridge
```

```
##      27      43      27      36      12      19      99
##
##
##  Nested Cross Validation averages for neural network :
##
##      deviance per event :
##     Uninformed   l/lasso feat l/lasso update
##          7.868          7.003          6.590
##
##      linear calibration coefficient :
##     Uninformed   l/lasso feat l/lasso update
##          0.485          0.629          0.907
##
##      average agreement (concordance) :
##     Uninformed   l/lasso feat l/lasso update
##          0.807          0.845          0.847
##
##  Cross validation informed neural network :
##
##      naive agreement :
##     Uninformed   l/lasso feat l/lasso update
##          0.921          0.884          0.857
```

```
#names(nested.cox.fit)
```

Before providing analysis results the output first reports sample size and since this is for a Cox regression, the number of events, followed by the number of predictors and the df (degrees of freedom) of the design matrix, as well as some information on "Tuning parameters". For the lasso model the tuning parameter that are not informed by the cross validation (CV) is the number of folds used in the CV and nested CV. When using CV to inform a stepwise procedure regression as described in JWHT (James, Witten, Hastie and Tibshirani, An Introduction to Statistical Learning with applications in R, 2nd ed., Springer, New York, 2021) the user should specify the maximum number of steps unless the number of predictors is small. We have found in practice that in general the lasso does better than the stepwise procedure so we do not present results here. (The tuned stepwise fits also take a long to run, possibly a part of the earlier motivation for the relaxed lasso model development.) One can also compare the performance of the lasso with that of a gradient boosting machine (GBM here implemented using the extreme gradient boosting and the *xgboost* package) by specifying doxgb=1, which we have not done in this call. We do though compare the fit with that of an Artificial Neural Network (ANN) model by specifying doann=1.

Next are the nested cross validation results. First are the per record (or per event in case of the Cox model) log-likelihoods which reflect the amount of information in each observation. Since we are not using large sample theory to base inferences we feel the per record are more intuitive. Next are the average number of model terms which reflect the complexity of the different models, even if in a naive sense, then the linear calibration coefficients obtained by regressing the predicteds on outcome, followed by the agreement statistics, here concordance. These nested cross validated concordances should be essentially unbiased for the given design, unlike the naive concordances where the same data are used to derive the model and calculate the concordances (see SRDM).

In addition to evaluating the CV informed relaxed lasso model using another layer of CV, the nested.glmnetr() function also runs cv.glmnetr() based upon the whole data set. Here we see, not unexpectedly, that the concordances estimated from the nested CV are slightly smaller than the concordances naively calculated using the original dataset. Depending on the data the nested CV and naive agreement measures, here concordance, can be very similar or disparate.

Following JWHT we provide information on the minimizing lasso models as well as the "1SE" models, which may be near to the minimizing lasso model fits, but of simpler nature. We though focus on the minimizing lasso fits recognizing that relaxed lasso and fully relaxed lasso fits generally provide models of simpler form while still optimizing a fit.

A summary for the CV fit can be produced by using the summary() function directly on a nested.glmnetr() output using the option *cvfit=TRUE*. Else one can also extract the CV fit by extracting the object$cv.glmnet.fit, where object is the output object obtained when running nested.glmnetr(). The plot() and predict() functions can be applied direclty to a nested.glmnetr() object without the *cvfit* option for futher evaluation or calculations for the CV model fit.

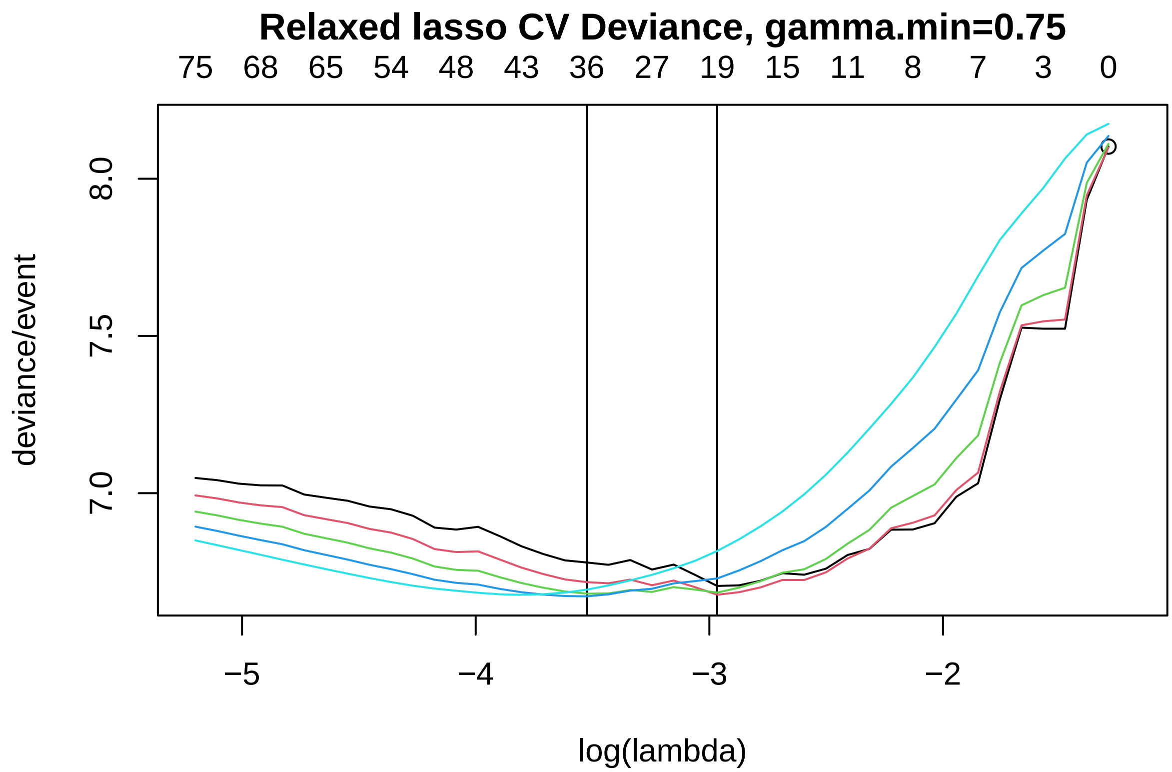
```
# Summary of the CV fit from a nested CV output
summary(nested.cox.fit, cvfit=TRUE)
```

```
##
##   The relaxed minimum is obtained for lambda = 0.0294621 , index = 25 and gamma = 0.75
##   with df (number of non-zero terms) = 36, average deviance = 6.671812 and beta =
##            X2            X3            X5            X8            X9
##  2.603455e-01  5.386785e-01 -5.013982e-01  2.462008e-01 -8.700892e-01
##           X10           X11           X12           X13           X14
## -5.018022e-01  8.041330e-02  2.337743e-01 -3.316142e-14  3.589660e-01
##           X18           X19           X20           X21           X22
##  7.581072e-01  1.979885e-01 -4.719132e-01 -6.272756e-01 -2.606001e-01
##           X23           X24           X25           X26           X27
##  4.553037e-01 -4.211458e-01 -7.368891e-01  3.628222e-02 -2.216800e-02
##           X30           X33           X39           X44           X50
## -1.229863e-01  4.093160e-02 -5.357801e-02 -4.101247e-02 -6.067097e-02
##           X55           X57           X58           X67           X70
##  4.578066e-02  2.548114e-02  1.720341e-02 -3.144855e-02  6.358778e-02
##           X73           X74           X75           X90           X98
##  3.766194e-02 -4.855725e-02 -2.098600e-02  4.033733e-02  1.996693e-02
##           X99
##  1.069094e-01
##
##   The fully relaxed (gamma=0) minimum is obtained for lambda = 0.05148586 and index = 19
##   with df (number of non-zero terms) = 18, average deviance = 6.704789 and beta =
##         X2         X3         X5         X8         X9        X10        X14
##  0.4464710  0.7814314 -0.5284294  0.3012989 -0.9963434 -0.6520494  0.5484414
##        X18        X19        X20        X21        X22        X23        X24
##  0.8407381  0.2609653 -0.5505213 -0.6993674 -0.3328967  0.5266075 -0.4782524
##        X25        X30        X70        X99
## -0.8360974 -0.2023634  0.1138980  0.1769053
##
##   The UNrelaxed (gamma=1) minimum is obtained for lambda = 0.022287 and index = 28
##   with df (number of non-zero terms) = 43, average deviance = 6.676892
##
##
##   Order coefficients entered into the lasso model (1st to last):
##  [1] "X18" "X21" "X25" "X23" "X9"  "X20" "X24" "X5"  "X22" "X10" "X19" "X3"
## [13] "X30" "X14" "X99" "X8"  "X2"  "X70" "X39" "X50" "X74" "X12" "X33" "X55"
## [25] "X73" "X44" "X67" "X90" "X26" "X57" "X75" "X98" "X11" "X27" "X58" "X78"
## [37] "X42" "X45" "X48" "X56" "X79" "X85" "X93"
```

Observe, the summary here is slightly different than obtained above running cv.glmnetr(). This is because the model is derived using a new call (instance) of the cv.glmnetr() function, and each CV uses by default a new random partitioning of the data.

```r
# Plot CV deviances from a nested CV output
plot(nested.cox.fit)
```

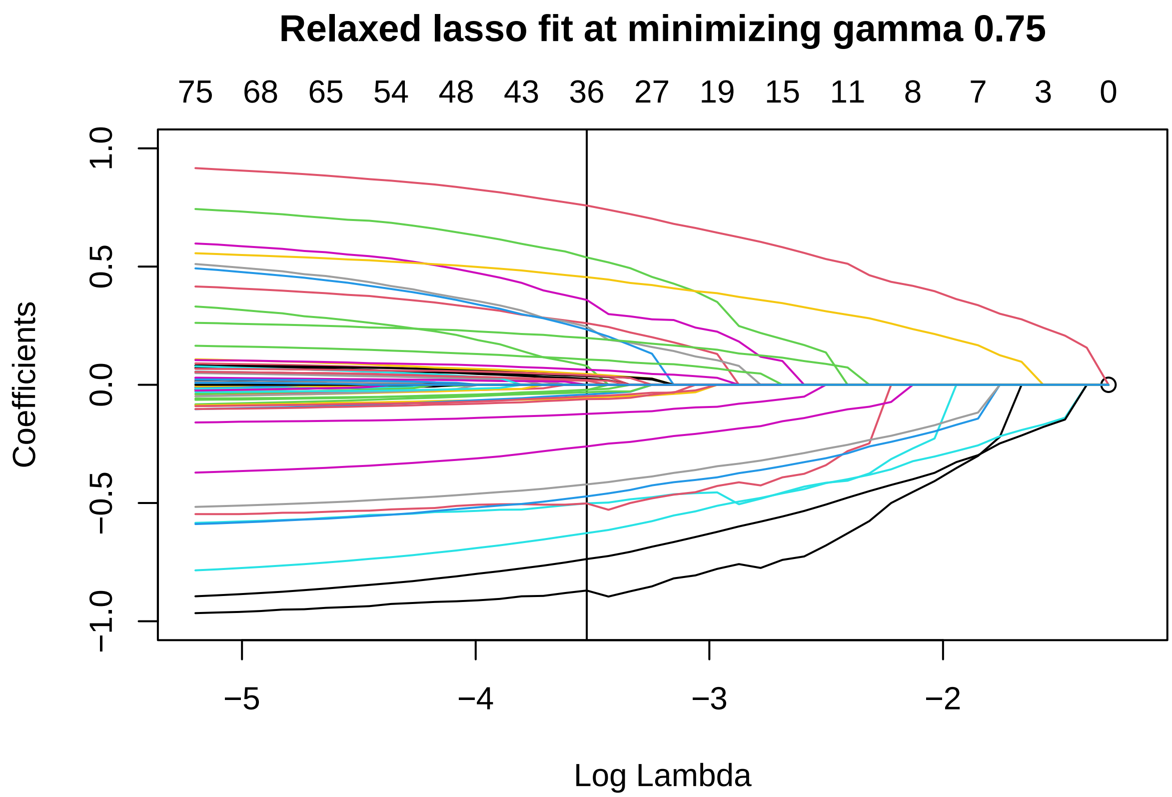
```
## min CV average deviance (max log likelihood) for
##   relaxed at log(lambda)  = -3.524, gamma.min = 0.75, df = 36
##   fully relaxed at log(lambda)   = -2.966, df = 19
##   fully penalized at log(lambda) = -3.803, df = 43
```



and

```r
# Plot coefficients from a nested CV output
plot(nested.cox.fit, coefs=TRUE)
```

```
## min CV average deviance (max log likelihood)
##  at log(lambda.min) = -3.525, gamma.min = 0.75, df = 36
```



Summarizing, the summary() function with the *cvfit*=TRUE option as well as the plot() and predict() functions for a nested.glmnetr() object are then essentially the same as those for a cv.glmnetr() output object. The summary() function without the *cvfit*=TRUE option, though, regards the evaluation of the cv.glmnetr() fit and is different.

```
# ... or use the predict() function on the CV fit embedded in the nested CV output
predict(nested.cox.fit)$beta
```

```
##
##  (lambda, gamma) pair minimizing CV average deviance is used

##            X2            X3            X5            X8            X9
##  2.603455e-01  5.386785e-01 -5.013982e-01  2.462008e-01 -8.700892e-01
##           X10           X11           X12           X13           X14
## -5.018022e-01  8.041330e-02  2.337743e-01 -3.316142e-14  3.589660e-01
##           X18           X19           X20           X21           X22
##  7.581072e-01  1.979885e-01 -4.719132e-01 -6.272756e-01 -2.606001e-01
##           X23           X24           X25           X26           X27
##  4.553037e-01 -4.211458e-01 -7.368891e-01  3.628222e-02 -2.216800e-02
##           X30           X33           X39           X44           X50
## -1.229863e-01  4.093160e-02 -5.357801e-02 -4.101247e-02 -6.067097e-02
##           X55           X57           X58           X67           X70
##  4.578066e-02  2.548114e-02  1.720341e-02 -3.144855e-02  6.358778e-02
##           X73           X74           X75           X90           X98
##  3.766194e-02 -4.855725e-02 -2.098600e-02  4.033733e-02  1.996693e-02
##           X99
##  1.069094e-01
```

Again, the plots and summary outputs from the nested.glmnetr() output are sightly different from what we saw above when summarizing the cv.glmnetr() output due to random data partitions for the CV folds.

Relaxed lasso fits or linear and lotistic model extensions

The glmnetr.simdata() can be used to obtain example data not only survival analyses but also for linear models and logistic models. The glmnetr.simdata() output object list contains not only xs for the predictor matrix, yt for time to event or censoring and event for event indication but also y_ for a normally distributed random variable for the linear model setting and yb for the logistic model setting. Below we show examples extracting and analyzing simulated data and for the linear model and logistic model structures. These show CV fits but nested CV fits can be performed similarly.

```
#=========================================================
# use the same simulated data output object from above, that is from the call
# simdata=glmnetr.simdata(nrows=1000, ncols=100, beta=NULL)
#
# extract linear regression model data
# xs = simdata$xs       # just as a comment as we did this above
yg = simdata$y_         # vector of Gaussian (normal) outcomes
# run a linear regression lasso model
cv.lin.fit = suppressWarnings(cv.glmnetr(xs,NULL,yg,NULL,family="gaussian",track=0))
summary(cv.lin.fit)
```

```
##
##   The relaxed minimum is obtained for lambda = 0.11524314 , index = 28 and gamma = 0.25
##   with df (number of non-zero terms) = 20, average deviance = 3.231872 and beta =
##             X2            X3            X5            X8            X9
##  3.324691e-01  1.126240e+00 -9.876594e-01  4.476292e-01 -2.475656e+00
##            X10           X12           X13           X14           X18
## -1.545063e+00  7.691733e-01 -9.568123e-15  2.962615e-01  1.252273e+00
##            X19           X20           X21           X22           X23
##  2.749915e-01 -7.778729e-01 -1.227520e+00 -3.522377e-01  8.755801e-01
##            X24           X25           X41           X90           X99
## -7.187621e-01 -1.480273e+00 -1.429797e-01  1.316714e-01  1.123756e-01
##
##   The fully relaxed (gamma=0) minimum is obtained for lambda = 0.11524314 and index = 28
##   with df (number of non-zero terms) = 19, average deviance = 3.236497 and beta =
##          X2         X3         X5         X8         X9        X10        X12
##  0.4159840  1.2337510 -1.0034780  0.5082731 -2.5178480 -1.6023040  0.8879799
##        X14        X18        X19        X20        X21        X22        X23
##  0.3737939  1.2820133  0.3013968 -0.8071176 -1.2490075 -0.3781211  0.9014142
##        X24        X25        X41        X90        X99
## -0.7495588 -1.5155377 -0.1774365  0.1603409  0.1370001
##
##   The UNrelaxed (gamma=1) minimum is obtained for lambda = 0.06594636 and index = 34
##   with df (number of non-zero terms) = 32, average deviance = 3.305662
##
##
##   Order coefficients entered into the lasso model (1st to last):
##  [1] "X21"  "X25"  "X18"  "X9"   "X23"  "X20"  "X24"  "X5"   "X10"  "X3"
## [11] "X22"  "X8"   "X19"  "X12"  "X90"  "X99"  "X2"   "X14"  "X41"  "X75"
## [21] "X30"  "X36"  "X50"  "X57"  "X46"  "X70"  "X77"  "X11"  "X38"  "X51"
```

```
## [31] "X100" "X29"
```

```
# plot(cv.lin.fit, coefs=TRUE)
#
# extract logistic regression model data
# xs = simdata$xs      # just as a comment as we did this above
yb = simdata$yb        # vector of binomial (0 or 1) outcomes
# run a logistic regression lasso model
cv.bin.fit = suppressWarnings(cv.glmnetr(xs,NULL,yb,NULL,family="binomial",track=0))
summary(cv.bin.fit)
```

```
##
##   The relaxed minimum is obtained for lambda = 0.03845953 , index = 15 and gamma = 0
##   with df (number of non-zero terms) = 11, average deviance = 0.878441 and beta =
##         X3         X5         X9        X10        X18        X20        X21
##  0.8243633 -1.1912389 -2.0156545 -1.2738819  1.0035533 -0.6603845 -0.8596552
##        X22        X23        X24        X25
## -0.2996174  0.6597314 -0.6021638 -1.0775306
##
##   The fully relaxed (gamma=0) minimum is obtained for lambda = 0.03845953 and index = 15
##   with df (number of non-zero terms) = 11, average deviance = 0.878441 and beta =
##         X3         X5         X9        X10        X18        X20        X21
##  0.8243633 -1.1912389 -2.0156545 -1.2738819  1.0035533 -0.6603845 -0.8596552
##        X22        X23        X24        X25
## -0.2996174  0.6597314 -0.6021638 -1.0775306
##
##   The UNrelaxed (gamma=1) minimum is obtained for lambda = 0.01259376 and index = 27
##   with df (number of non-zero terms) = 35, average deviance = 0.909008
##
##
##   Order coefficients entered into the lasso model (1st to last):
##  [1] "X18" "X21" "X25" "X9"  "X20" "X23" "X24" "X5"  "X3"  "X10" "X22" "X37"
## [13] "X26" "X62" "X84" "X2"  "X35" "X82" "X29" "X77" "X80" "X11" "X19" "X56"
## [25] "X99" "X58" "X59" "X14" "X69" "X79" "X86" "X6"  "X60" "X73" "X81"
```

```
# plot(cv.bin.fit, coefs=TRUE)
```

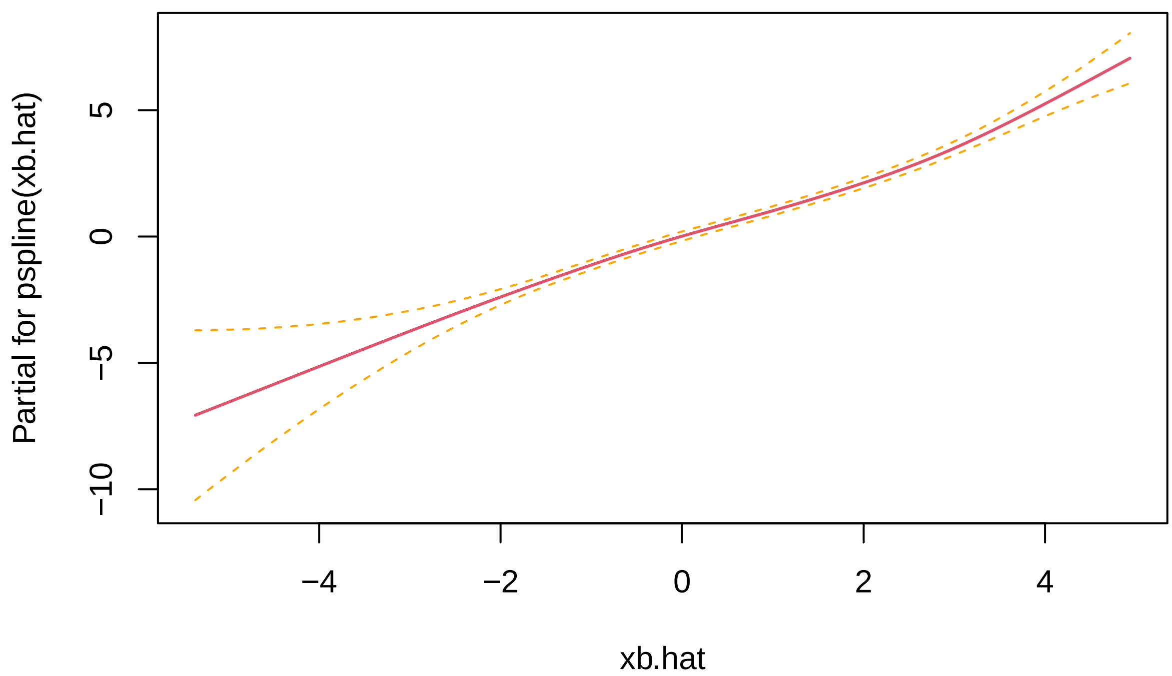
Further model assessment

One can also fit a spline to the predicteds obtained form the predict() functions. This may help to understand nonlinearities in the predicteds, but may also give inflated hazard ratios.

```
# Get predicteds from CV relaxed lasso model embedded in nested CV outputs & Plot
xb.hat = predict( object=cv.cox.fit , xs_new=xs, lam=NULL, gam=NULL, comment=FALSE)
# describe the distribution of xb.hat
round(1000*quantile(xb.hat,c(0.01,0.05,0.1,0.25,0.5,0.75,0.90,0.95,0.99)))/1000
```

```
##     1%     5%    10%    25%    50%    75%    90%    95%    99%
## -4.131 -2.887 -2.230 -1.165  0.082  1.267  2.266  2.982  3.874
```

```r
# Fit a spline to xb.hat uisng coxph, and plot
fit1 = coxph(Surv(y_, event) ~ pspline(xb.hat))
termplot(fit1,term=1,se=TRUE)
```



From this spline fit we see the predicteds are approximately linear with the log hazard ratio.