

# An Overview of glmnet

Walter K. Kremers, Mayo Clinic, Rochester MN

30 December 2023

## The Package

The `nested.glmnet()` function of the ‘glmnet’ package allows the user to fit multiple machine learning models on a common dataset with a single function call allowing an efficient comparison of different modeling approaches. Additionally this function uses cross validation to estimate model performances for these different modeling approaches. As most of these machine learning models choose hyperparameters informed by a cross validation or some sort of out of bag (OOB) performance measure, the `nested.glmnet()` function provides model performance estimates based upon either a nested cross validation or analogous approach. Measures of model performance include concordances for survival time and binomial outcomes and R-squares for quantitative numeric outcomes, as well as deviances and linear calibration coefficients. Too often one sees performance reports including things like sensitivity, specificity or F1 scores in absence of any consideration of calibration. Whereas linear calibration does not exhaust the needs of calibration considerations, it does provide a first high level insight. As the purpose of the function is to not only describe performance but to derive the models, each of the fitted models as well as performance measures are stored in a single output object.

The `nested.glmnet()` function fits cross validation informed Relaxed lasso, Artificial Neural Network (ANN), gradient boosting machine (‘xgboost’), Random Forest (‘RandomForestSRC’), Recursive Partitioning (‘RPART’) and step wise regression models. As run times may be long, the user specifies which of these models to fit. By default only the lasso model suite is fit, including the (standard) lasso, relaxed lasso, fully relaxed lasso ( $\gamma=0$ ) and the ridge regression model. (The program was originally written to simply compare the lasso and stepwise regression models and thus this inclusion of the lasso by default, as well as the program name.) By default model performances are calculated using cross validation but if the goal is to only fit the models this can be done using the option `do_ncv=0`.

As with the ‘glmnet’ package, tabular and graphical summaries can be generated using the `summary` and `plot` functions. Use of the ‘glmnet’ has many similarities to the ‘glmnet’ package and the user may benefit by a review of the documentation for 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.

For some datasets, for example when the design matrix is not of full rank, ‘glmnet’ may have very long run times when fitting the relaxed lasso model, from our experience when fitting Cox models on data with many predictors and many patients, making it difficult to get solutions from either `glmnet()` or `cv.glmnet()`. This may be remedied with the ‘`path=TRUE`’ option when calling `cv.glmnet()`. This option is not described in the ‘glmnet’ Reference Manual but is described in the ‘glmnet’ “The Relaxed Lasso” vignette. In the ‘glmnet’ package we provide a similar workaround when fitting the non penalized relaxed model where  $\gamma=0$ .

When 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.

## Data requirements

The basic data elements for input to the *glmnet* analysis programs are similar to those of *glmnet* and include 1) a matrix of predictors and 2) an outcome variable in vector form. For the different machine learning modeling approaches the package is set up to model generalizations of the proportional hazards Cox survival model, the “binomial” outcome logistic model and linear regression for well behaved numerical outcomes treated as if gaussian in distribution. When fitting 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.

The input vectors may optionally be specified as column matrices (with only one column each) in which case the column name will be kept and expressed in the model summaries.

## An example dataset

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

The code

```
# 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(116291949)  
simdata=glmnet.simdata(nrows=1000, ncols=100, beta=NULL)
```

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

```
# 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

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

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

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

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

```
# 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  0  0  1  0  1  0.1513225 -0.4034383  0.35250844
## [2,]  1  0  0  0  1  0  0  1  0  0  0  0 -1.1610480  0.5533030  0.14578868
## [3,]  1  0  0  1  0  0  1  0  0  0  0  0 -0.3292269  0.3086399 -0.48443836
## [4,]  1  1  0  0  0  0  0  0  0  1  0  0  2.0635214 -0.5500741 -0.02173104
## [5,]  1  0  0  0  1  0  0  1  0  0  0  0  0.3905722 -0.6836452 -0.37643201
## [6,]  1  0  1  0  0  0  0  0  1  0  0  0 -0.2397597  1.6909447  0.49599945
## [7,]  1  0  1  0  0  0  0  1  0  0  0  0 -0.5592424  0.2314638 -0.53198341
## [8,]  1  0  0  1  0  0  0  0  0  0  1  0 -1.0050514  0.5319574  0.54287646
## [9,]  1  0  0  1  0  0  0  0  0  0  1  0  1.2548034  0.8213164  0.17067691
## [10,] 1  0  0  0  1  0  0  0  1  0  0  0 -0.3079151 -0.6105910 -0.88711869
```

## Performance of cross validation (CV) informed relaxed lasso model 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> ). We demonstrate the model performance evaluation by nested cross validation first for the lasso models with the evaluation of other machine learning models being similar. For this performance evaluation we use the `nested.glmnet()` function which first fits lasso models based upon all data and then performs the cross validation for calculation of concordances or R-squares, deviances and linear calibration summaries.

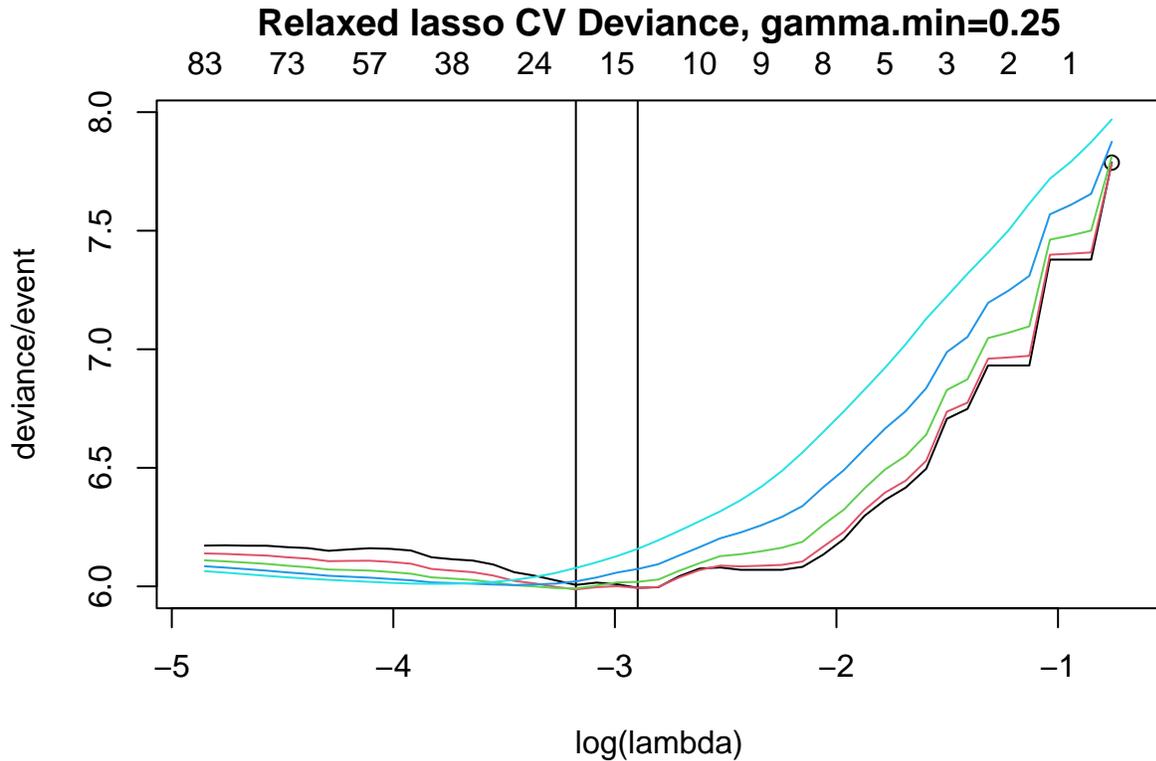
```
set.seed(465783345)
nested.cox.fit = suppressWarnings( nested.glmnet(xs, NULL, y_, event, family="cox",
                                               dostep=1, folds_n=10, track=0) )
```

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 “wrap” the call to `nested.glmnet()` within the `suppressWarnings()` function to avoid excessive warning messages in this vignette. The first term in the call to `nested.glmnet()`, `xs`, is the design matrix for predictors. The second input term, here `NULL`, is for the start time in case the (start, stop) time data setup is used in a Cox survival model fit. 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 fourth 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 fourth term would be `NULL` for either linear or logistic regression. 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 inspect the average cross validation deviance using the plot function.

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

```
## min CV average deviance (max log likelihood) for
## relaxed at log(lambda) = -3.176, gamma.min = 0.25, df = 19
## fully relaxed at log(lambda) = -2.897, df = 14
## fully penalized at log(lambda) = -3.827, df = 44
```



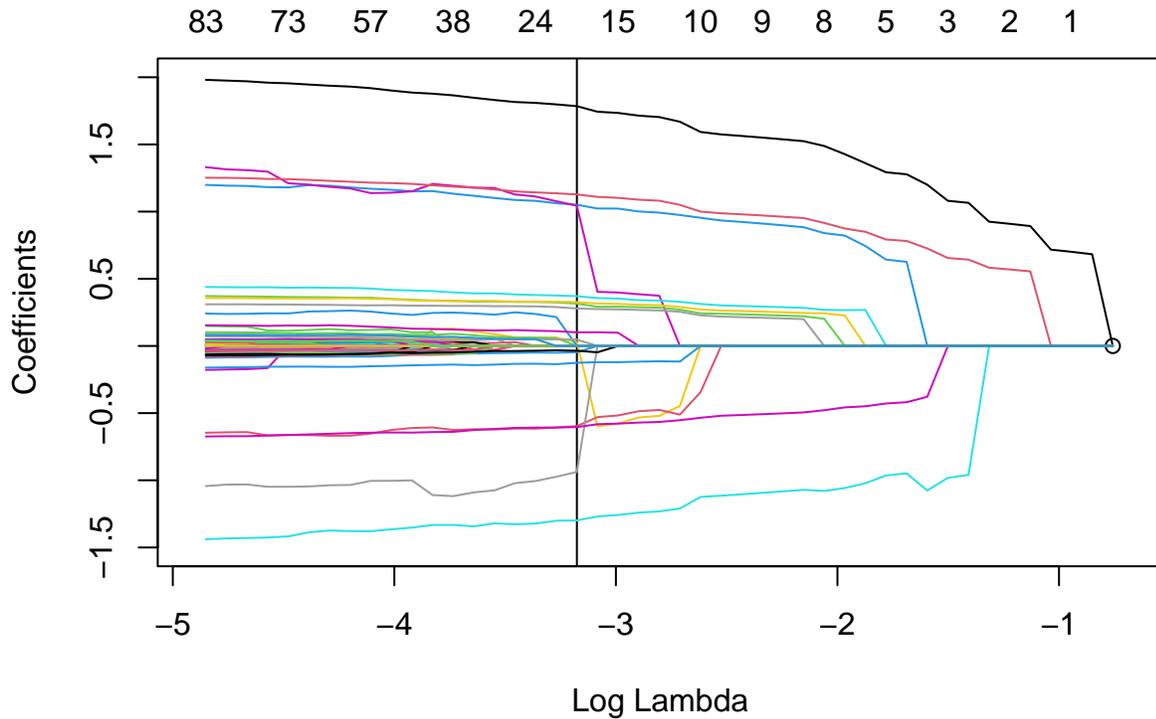
In that to maximize the log-likelihood is to minimize deviance we inspect these curves for a minimum. The minimizing lambda is indicated by the left most vertical line, here about  $\log(\lambda) = -3.18$ . The minimizing gamma is 0.25 and described in the title. Whereas there is no legend here for gamma, when non-zero coefficients start to enter the model as the penalty is reduced, here shown at the right, deviances 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  $\lambda=0.25$  the deviance is hardly distinguishable for gamma ranging from 0.5 to 1. More relevant we see that the fully unpenalized lasso model fits ( $\gamma=0$ ) shown in a black line with a black circle at the largest lambda, achieves a minimal deviance at about -2.9, and highlighted by the right most vertical line. The minimizing deviance for the fully relaxed lasso model is “nearly” that of the relaxed lasso model tuning for both lambda and gamma.

A plot depicting model fits as a function of lambda is given in the next figure.

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

```
## min CV average deviance (max log likelihood)
## at log(lambda.min) = -3.176, gamma.min = 0.25, df = 19
```

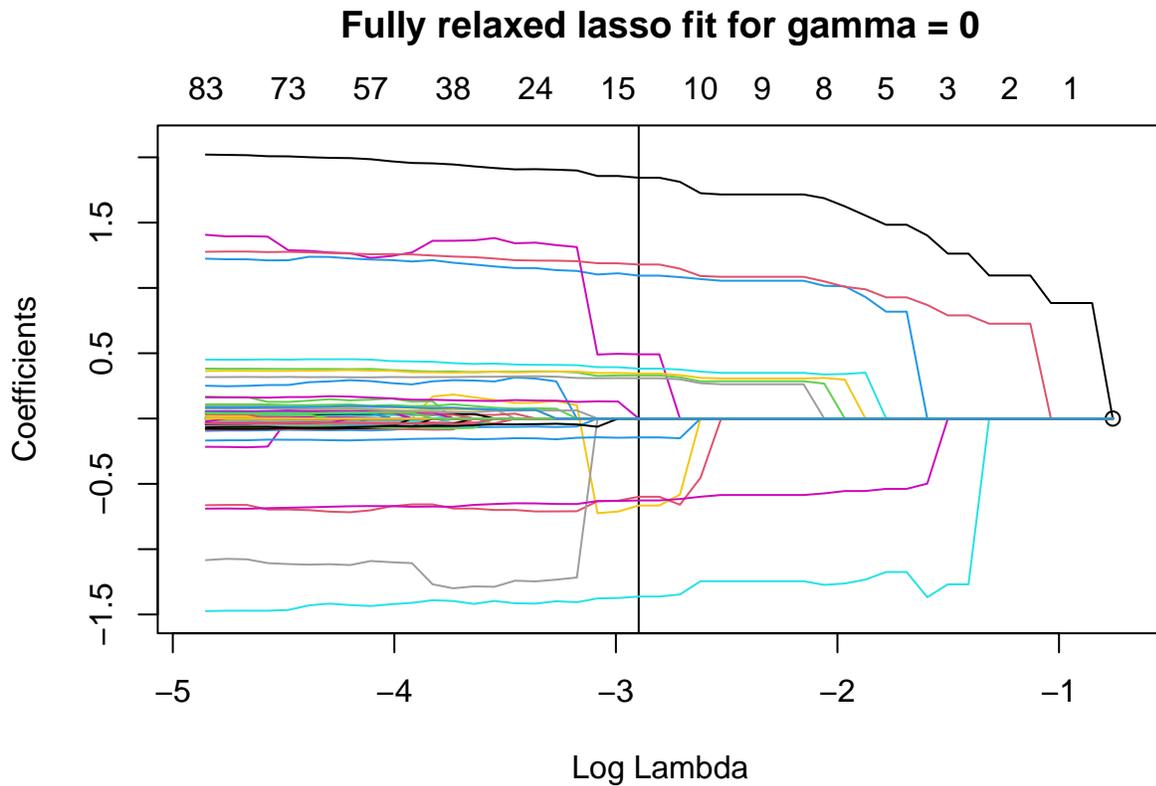
## Relaxed lasso fit at minimizing gamma 0.25



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=0.25 as noted in the tile, and the minimizing lambda indicated by the vertical line. 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(nested.cox.fit, coefs=TRUE, gam=0)
```

```
## Fully relaxed min CV average deviance (max log likelihood)  
## at log(lambda.min) = -2.897, df = 14
```



In addition to simply showing how the coefficients change as the lambda penalty is decreased, this plot shows how the coefficients change for the un-penalized (fully relaxed) model with  $\gamma=0$  as lambda decreases. In particular we see the coefficients become slightly larger in magnitude as the lambda penalty decreases and also as additional terms come into the model. 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, as too indicated in the deviance plot, the number of model non-zero coefficients, 14, to be substantially less than the 19 from the relaxed lasso fit and the 44 from the fully penalized lasso fit.

As usual with R functions and packages we use the summary function to describe output. Here the summary function displays a brief summary of the input data and program options before proceeding to describe model performances. The data summary includes sample size, number of events, number of candidate model predictors, degrees of freedom in these predictors as well as average deviance. Model performances are displayed for the different lasso models, e.g. standard, relaxed, fully relaxed as well as the ridge regression and stepwise regression models.

Hyperparameters considered for the lasso models include both the minimizing lambdas as well as the “1se” or “one standard deviation” lambdas for the relaxed lasso fit informed by CV. Hyperparameters considered for stepwise regression were degrees of freedom (df) and p, the p-value for entry into the regression equation, as discussed by JWHT (James, Witten, Hastie and Tibshirani, *An Introduction to Statistical Learning with applications in R*, 2nd ed., Springer, New York, 2021). Performance measures include deviance, linear calibration coefficients and measures of agreement, here for the Cox model framework concordance. Additionally there are the deviance and agreement from the whole sample. Observe how the deviances are much larger for the whole sample than for the deviances for the leave out samples in the (outer) cross validation. This is because the risk sets are larger for the whole sample leading to larger numbers derived at each event time.

```
# Summarize relaxed lasso model performance informed by cross validation
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"      "698"          "100"          "94"
## null.dev/events
##      "12.43"
##
## Tuning parameters for models :
##      folds_n stratified      limit      fine      ties      method      steps_n
##      "10"      "1"          "1"          "0"      "efron"      "loglik"      "100"
##
## Average Null Deviance for leave out folds in outer loop :
##      7.859
##
##
## Nested Cross Validation averages for LASSO (1se and min), Relaxed LASSO, and gamma=0 LASSO :
##
##      deviance per event :
##      1se      min      1seR      minR 1seR.GO minR.GO      ridge
##      5.971  5.924  5.944  5.920  5.937  5.929  6.095
##
##      deviance per event (linearly calibrated) :
##      1se      min      1seR      minR 1seR.GO minR.GO      ridge
##      5.901  5.899  5.909  5.896  5.914  5.895  5.998
##
##      number of nonzero model terms :
##      1se      min      1seR      minR 1seR.GO minR.GO
##      22.6  47.2  16.3  20.0  11.9  15.3
##
##      linear calibration coefficient :
##      1se      min      1seR      minR 1seR.GO minR.GO      ridge
##      1.248  1.083  1.150  1.006  0.982  0.963  1.313
##
##      agreement (concordance) :
##      1se      min      1seR      minR 1seR.GO minR.GO      ridge
##      0.874  0.873  0.874  0.875  0.873  0.874  0.866
##
## Naive deviance for cross validation informed LASSO :
##      1se      min      1seR      minR 1seR.GO minR.GO      ridge
##      10.47547 10.32173 10.47907 10.30440 10.29365 10.23326 10.29460
##
## Number of non-zero terms in cross validation informed LASSO :
##      1se      min      1seR      minR 1seR.GO minR.GO      ridge
##      19      44      14      19      10      14      99
##
## Naive agreement for cross validation informed LASSO :
##      1se      min      1seR      minR 1seR.GO minR.GO      ridge
##      0.874  0.879  0.873  0.879  0.879  0.881  0.882
##
##
## Nested Cross Validation STEPWISE regression model (df):
##      Average linear calibration coefficient: 0.973
##      Average deviance : 5.863

```

```

##      Average model df : 14.5
##      Concordance      : 0.878
## Naive concordance based upon the same (all) data as model derivation (df): 0.878
##      Model df 15
##
## Nested Cross Validation STEPWISE regression model (p):
##      Average linear calibration coefficient p : 0.973
##      Average deviance : 5.854
##      Average model p  : 0.013
##      Average model df : 15
##      Concordance      : 0.879
## Naive concordance based upon the same (all) data as model derivation (p): 0.878
##      Model df 15

```

From this output we also see the number of non-zero coefficients in the different models, reflecting model complexity, along with the linear calibration coefficients obtained by regressing the outcome on the predicted.

In addition to evaluating the CV informed relaxed lasso model using another layer of CV, the `nested.glmnet()` function also calculates model performances based upon all data for the model based upon all data. 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.

A summary of the actual lasso model fit can be gotten by using the `cvfit=1` option in the `summary()` call.

```

# Summarize relaxed lasso model fit informed by cross validation
summary(nested.cox.fit, cvfit=1)

```

```

##
## The relaxed minimum is obtained for lambda = 0.04174656 , index = 27 and gamma = 0.25
## with df (number of non-zero terms) = 19, average deviance = 5.987085 and beta =
##      X4      X5      X7      X10     X14
## 1.050952e+00 -1.298945e+00 2.912949e-02 -5.966936e-01 1.043615e+00
##      X15     X16     X18     X19     X20
## -1.719568e-16 -9.378331e-01 1.127645e+00 3.149391e-01 -1.253247e-01
##      X21     X22     X23     X24     X25
## 3.707304e-01 -6.039354e-01 3.263490e-01 2.789101e-01 1.784703e+00
##      X38     X60     X88     X97
## 1.011793e-01 -4.634282e-02 4.740086e-02 -3.554883e-02
##
## The fully relaxed (gamma=0) minimum is obtained for lambda = 0.0551865 and index = 24
## with df (number of non-zero terms) = 13, average deviance = 5.993744 and beta =
##      X4      X5      X7      X10     X14     X18     X19
## 1.0941352 -1.3627747 -0.6659735 -0.5988756 0.4916291 1.1792301 0.3302211
##      X20     X21     X22     X23     X24     X25
## -0.1444546 0.3820778 -0.6270995 0.3448224 0.3085746 1.8435742
##
## The UNrelaxed (gamma=1) minimum is obtained for lambda = 0.02176669 and index = 34
## with df (number of non-zero terms) = 44, average deviance = 6.010734
##
##
## Order coefficients entered into the lasso model (1st to last):
## [1] "X25" "X18" "X5" "X22" "X4" "X21" "X23" "X19" "X24" "X10"
## [11] "X7" "X20" "X14" "X38" "X97" "X16" "X60" "X88" "X12" "X43"
## [21] "X71" "X100" "X34" "X32" "X50" "X58" "X41" "X49" "X64" "X84"

```

```
## [31] "X91" "X98" "X39" "X40" "X66" "X73" "X74" "X11" "X61" "X69"
## [41] "X70" "X96" "X63" "X77"
```

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, which is done by not specifying a predictor matrix. If one specifies a new design matrix xs\_new, then the predicted xs\_new\*beta are generated. In contrast to the summary function which simply displays coefficients, the predict function provides an output object in vector form (actually 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 here.

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

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

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

```
##           X4           X5           X7           X10           X14
## 1.050952e+00 -1.298945e+00 2.912949e-02 -5.966936e-01 1.043615e+00
##           X15           X16           X18           X19           X20
## -1.719568e-16 -9.378331e-01 1.127645e+00 3.149391e-01 -1.253247e-01
##           X21           X22           X23           X24           X25
## 3.707304e-01 -6.039354e-01 3.263490e-01 2.789101e-01 1.784703e+00
##           X38           X60           X88           X97
## 1.011793e-01 -4.634282e-02 4.740086e-02 -3.554883e-02
```

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

```
##           X1           X2           X3           X4           X5
## 0.000000e+00 0.000000e+00 0.000000e+00 1.050952e+00 -1.298945e+00
##           X6           X7           X8           X9           X10
## 0.000000e+00 2.912949e-02 0.000000e+00 0.000000e+00 -5.966936e-01
```

```

##          X11          X12          X13          X14          X15
## 0.000000e+00 0.000000e+00 0.000000e+00 1.043615e+00 -1.719568e-16
##          X16          X17          X18          X19          X20
## -9.378331e-01 0.000000e+00 1.127645e+00 3.149391e-01 -1.253247e-01
##          X21          X22          X23          X24          X25
## 3.707304e-01 -6.039354e-01 3.263490e-01 2.789101e-01 1.784703e+00
##          X26          X27          X28          X29          X30
## 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
##          X31          X32          X33          X34          X35
## 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
##          X36          X37          X38          X39          X40
## 0.000000e+00 0.000000e+00 1.011793e-01 0.000000e+00 0.000000e+00
##          X41          X42          X43          X44          X45
## 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
##          X46          X47          X48          X49          X50
## 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
##          X51          X52          X53          X54          X55
## 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
##          X56          X57          X58          X59          X60
## 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 -4.634282e-02
##          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 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
##          X71          X72          X73          X74          X75
## 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
##          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 4.740086e-02 0.000000e+00 0.000000e+00
##          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 -3.554883e-02 0.000000e+00 0.000000e+00 0.000000e+00

```

```

# Get the predicted values (linear predictors) for the original data set
predicted = predict(nested.cox.fit, xs)

```

```

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

```

```

# Print out the first few predicted values
predicted[1:20]

```

```

## [1] -0.6446191 -3.4901590 4.3166516 1.3425973 -0.1500069 1.2901788
## [7] -3.8608813 0.3456247 6.1151657 1.5431362 1.0919012 -1.7379752
## [13] 0.7941607 2.7537587 -0.6522066 0.5313555 0.8459184 3.3600472
## [19] -2.4937645 1.9657998

```

## Nested cross validation for multiple machine learning models

Here we evaluate multiple machine learning models, in particular the lasso, XGB, random forest and neural network models. For this example too we perform an analysis for the generalizations of linear regression in contrast to the Cox model in the last example. The `glmnet.simdata()` function used above actually creates an 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.

```
# Nested cross validation evaluated machine learning model suite and gaussian errors
# use the same simulated data output object from above, that is from the call
# simdata=glmnet.simdata(nrows=1000, ncols=100, beta=NULL)
#
# extract linear regression model data
yg = simdata$y_ # vector of Gaussian (normal) outcomes
# Get the ML model fits
nested.gau.fit = suppressWarnings(nested.glmnet(xs=NULL,yg=NULL,family="gaussian",
  dolasso=1, doxgb=1, dorf=1, doann=1, ensemble=c(1,0,0,0, 0,1,0,1), folds_n=10,
  seed=219301029, track=0))
summary(nested.gau.fit)
```

```
##
## Sample information including number of records, number of columns in
## design (predictor, X) matrix, and df (rank) of design matrix:
##   family      n  xs.columns  xs.df null.dev/obs
## "gaussian"  "1000"    "100"    "94"    "8.09"
##
## Tuning parameters for models :
##   folds_n stratified  limit  fine  ties
##   "10"    "1"        "1"   "0"  "efron"
##
## 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
##
## Average Null Deviance for leave out folds in outer loop :
##   8.067
##
##
## Nested Cross Validation averages for LASSO (1se and min), Relaxed LASSO, and gamma=0 LASSO :
##
##   deviance per record :
##   1se  min  1seR  minR 1seR.GO minR.GO  ridge
##   1.161  1.123  1.138  1.128  1.132  1.139  1.213
##
##   deviance per record (linearly calibrated) :
##   1se  min  1seR  minR 1seR.GO minR.GO  ridge
##   1.127  1.106  1.118  1.114  1.120  1.127  1.140
##
##   number of nonzero model terms :
##   1se  min  1seR  minR 1seR.GO minR.GO
```

```

##      30.1      60.5      21.2      28.9      17.5      22.9
##
##      linear calibration coefficient :
##      1se      min      1seR      minR 1seR.GO minR.GO      ridge
##      1.056    1.018    1.036    1.013  0.996    0.993    1.097
##
##      agreement (r-square) :
##      1se      min      1seR      minR 1seR.GO minR.GO      ridge
##      0.859    0.861    0.860    0.860  0.859    0.859    0.857
##
##      Naive deviance for cross validation informed LASSO :
##      1se      min      1seR      minR 1seR.GO minR.GO      ridge
##      1.0547535 0.9760810 1.0547535 0.9760810 0.9762544 0.9423096 0.9659641
##
##      Number of non-zero terms in cross validation informed LASSO :
##      1se      min      1seR      minR 1seR.GO minR.GO      ridge
##      36      66      36      66      12      26      99
##
##      Naive agreement for cross validation informed LASSO :
##      1se      min      1seR      minR 1seR.GO minR.GO      ridge
##      0.871    0.880    0.871    0.880  0.879    0.884    0.884
##
##
##      Nested Cross Validation averages for XGBoost model :
##
##      deviance per record :
##      Simple Feature  Offset  Tuned Feature  Offset
##      2.466    1.445    1.153    1.518    1.214    1.129
##
##      linear calibration coefficient :
##      Simple Feature  Offset  Tuned Feature  Offset
##      1.129    0.997    0.987    1.080    1.032    0.993
##
##      agreement (r-square) :
##      Simple Feature  Offset  Tuned Feature  Offset
##      0.705    0.821    0.857    0.818    0.851    0.860
##
##      Naive agreement for cross validation informed XGBoost model :
##      Simple Feature  Offset  Tuned Feature  Offset
##      0.999    1.088    0.895    0.946    0.894    0.880
##
##
##      Nested Cross Validation averages for Random Forest :
##
##      deviance per record :
##      None Feature  Offset
##      2.332    1.231    1.138
##
##      average number of variables randomly selected for the RF :
##      None Feature  Offset
##      58.0    59.9    32.5
##
##      linear calibration coefficient :
##      None Feature  Offset

```

```

## 1.218 1.030 0.991
##
## agreement (r-square) :
## None Feature Offset
## 0.737 0.848 0.859
##
## Naive Random Forest agreement (r-square) :
## Standard Feature Offset
## 0.950 0.967 0.965
##
##
## Nested Cross Validation averages for neural network :
##
## deviance per record :
## Uninformed 1/lasso feat 1/lasso update
## 2.602 1.271 1.147
##
## linear calibration coefficient :
## Uninformed 1/lasso feat 1/lasso update
## 0.949 0.981 0.994
##
## agreement (r-square) :
## Uninformed 1/lasso feat 1/lasso update
## 0.683 0.843 0.860
##
## Cross validation informed neural network :
##
## naive agreement :
## Uninformed 1/lasso feat 1/lasso update
## 0.959 0.927 0.880

```

Here we see a set of machine learning models evaluated together. Information displayed is similar to what we saw before with the main difference being that for many ML models there are no “number of non-zero terms” like for the lasso. All evaluations are based upon the same folds for the outer loop of the cross validation. Those models informed by cross validation in identification of hyperparameters, i.e. lasso, neural network and stepwise, use the same folds in the inner cross validation making the comparisons of model performance between models more stable. For the models based upon other random splittings, i.e. XGB, random forest and RPART, the same seed is set using `set.seed()` before each model call facilitating replicability of results.

The individual model fits are all captured in the `nested.glmnetr()` output object with names like `cv_glmnet_fit`, `xgb.simple.fit`, `xgb.tuned.fit`, `rf.fit`, `cv.stepreg.fit` and `ann_fit_X` with X corresponding to the ensemble designation. `cv_glmnet_fit` has a similar yet different format to that of `cv.glmnet()` by including further fit information. The XGB outputs are direct outputs from ‘xgboost’ `xgb.train()`. The `rf.fit` object contains the output from `rfsrc()` in the object `rf.fit$rf_tuned` along with other information used for tuning. The `ann_fit_X` objects are derived using the R ‘torch’ package and take on their own format for logistical reasons. See the ‘Using `ann_tab_cv`’ vignette. Cross validation information from the individual outer folds are contained in datasets like `xx.devian.cv`, `xx.lincal.cv`, `xx.agree.cv` for further processing by the `summary()` function or by the user. For example

```

# Manually calculate CV R_square for lasso models
corr.cv = nested.gau.fit$lasso.agree.cv
avecorr = colMeans(corr.cv)
R_square = avecorr ^2
R_square

```

```
##          1se          min          1seR          minR          1seR.GO          minR.GO          ridge
## 0.8586666 0.8612704 0.8597374 0.8602448 0.8594145 0.8587198 0.8569730
```

These numbers are consistent with the output from the `summary()` call.

In this and the previous examples using `nested.glmnetr()` we specified values for `seed`. This assures that the user can test the program with their own installation and get the same results. Typically in practice one can leave the seed unspecified and the program will generate its own seeds and store these in the output object (`object$seed`) for future reference. One should be cautious of using `set.seed()` in one's own code as this too will effect the pseudo randomness used in the ML calculations and could unwantingly yield identical results when pseudo independent runs are intended.

## Further model assessment

Further model assessment can be performed based upon the predicted values from the `predict` functions. For example, one can model the outcomes based upon a spline for the  $X \cdot \beta$  hats from the predicted values. This may help to understand potential nonlinearities in the model, but may also give inflated hazard ratios.

```
# Get predicted values from CV relaxed lasso model embedded in nested CV outputs & Plot
xb.hat = predict(object=nested.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
```

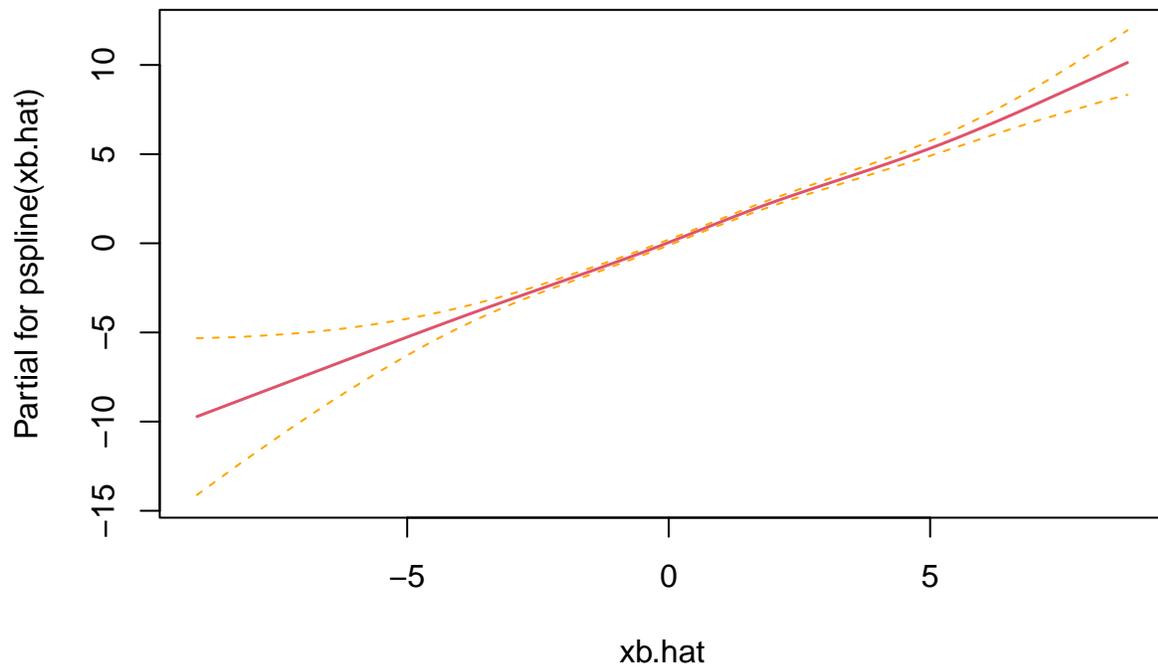
```
##          1%          5%          10%          25%          50%          75%          90%          95%          99%
## -5.839 -4.122 -3.233 -1.804 -0.070  1.578  3.188  3.989  5.449
```

```
# Fit a spline to xb.hat using coxph, and plot
fit1 = coxph(Surv(y_, event) ~ pspline(xb.hat))
summary(fit1)
```

```
## Call:
## coxph(formula = Surv(y_, event) ~ pspline(xb.hat))
##
##      n = 1000, number of events = 698
##
##              coef se(coef) se2      Chisq  DF  p
## pspline(xb.hat), linear 1.073 0.03335 0.03335 1034.93 1.00 4.6e-227
## pspline(xb.hat), nonlin          3.77 3.04 2.9e-01
##
##              exp(coef) exp(-coef) lower .95 upper .95
## ps(xb.hat)3  7.221e+00  1.385e-01 7.047e-01 7.400e+01
## ps(xb.hat)4  5.215e+01  1.918e-02 8.007e-01 3.396e+03
## ps(xb.hat)5  3.764e+02  2.657e-03 1.508e+00 9.393e+04
## ps(xb.hat)6  2.664e+03  3.754e-04 5.104e+00 1.390e+06
## ps(xb.hat)7  1.634e+04  6.121e-05 2.699e+01 9.886e+06
## ps(xb.hat)8  1.036e+05  9.656e-06 1.791e+02 5.990e+07
## ps(xb.hat)9  9.080e+05  1.101e-06 1.561e+03 5.283e+08
## ps(xb.hat)10 4.941e+06  2.024e-07 8.443e+03 2.892e+09
## ps(xb.hat)11 2.882e+07  3.470e-08 4.875e+04 1.704e+10
## ps(xb.hat)12 2.757e+08  3.627e-09 4.437e+05 1.713e+11
## ps(xb.hat)13 3.015e+09  3.317e-10 4.081e+06 2.228e+12
```

```
## ps(xb.hat)14 3.389e+10 2.950e-11 2.878e+07 3.991e+13
##
## Iterations: 4 outer, 16 Newton-Raphson
## Theta= 0.7383276
## Degrees of freedom for terms= 4
## Concordance= 0.879 (se = 0.005 )
## Likelihood ratio test= 1494 on 4.04 df, p=<2e-16
```

```
termplot(fit1,term=1,se=TRUE)
```



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