An Overview of glmnetr

Walter K. Kremers, Mayo Clinic, Rochester MN

12 September 2026

The Package

The nested.glmnetr() function of the 'glmnetr' package allows the user to fit multiple machine learning models on a common data set with a single function call allowing an efficient comparison of different modeling approaches. Additionally this function uses cross validation (CV) or a boostrap method to estimate model performances for these different modeling approaches from the hold out (out of bag) sample partitions. As most of these machine learning models choose hyperparameters informed by a cross validation or some sort, or out of bag (OOB) performance measure, the nested.glmnetr() function provides model performance estimates based upon either a nested cross validation (NCV), bootstraping or analogous approach. Measures of model performance include concordances for survival time and binomial outcomes, R-squares for quantitative numeric outcomes, as well as deviance ratios, i.e. 1-deviance(model)/deviance(nullmodel), 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. In addition to providing linear calibration coefficient estimates, we also show spline fits on the hold out datasets of the nested cross validation outer folds. The nested.glmnetr() function also fits the respective models on the whole dataset. Performance measures and fitted models based upon the whole dataset are stored in a single output object.

The nested.glmnetr() function fits cross validation informed lasso (or more generally elastic net), relaxed lasso, ridge, gradient boosting machine ('xgboost'), Random Forest ('RandomForestSRC'), Oblique Random Forest ('aorsf'), Artificial Neural Network (ANN), Recursive Partitioning and Regression Trees ('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 models. (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 resample=0.

As with the 'glmnet' package, tabular and graphical summaries can be generated using the summary and plot functions. Use of the 'glmnetr' package 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 data sets, 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(). In the 'glmnetr' package we always take an approach like that of path=TRUE.

When fitting not a relaxed lasso model but an elastic-net model, 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.

Installing glmnetr

Installing glmnetr is much like installing other R packages, but with some small wrinkles. As usual one submits

```
install.packages( 'glmnetr' )
```

and then loads the package from the library

```
library( glmnetr )
```

Additionally, when loading the package for the first time one may be prompted to load further software needed to run the torch library for the neural network models, for example as in

```
> library( glmnetr )
i Additional software needs to be downloaded and installed for torch to work correctly.
Do you want to continue? (Yes/no/cancel)
```

where response Yes should allow neural network model fitting. A no answer will not install torch, and if not already installed then attempting to run the neural network models will lead to crashes.

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 in vector form. For the different machine learning modeling approaches the package is set up to model generalizations of the Cox proportional hazards survival model, the "binomial" outcome logistic model and linear regression with independent identically distributed errors amenable to being treated as if "gaussian". When fitting the Cox model the outcome model variable is interpreted as the "time" variable, 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 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 data set

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

```
# Simulate data for use in an example for relaxed lasso fit of survival data
# First, optionally, assign a seed for random number generation to get replicable results
set.seed(116291949)
simdata=glmnetr.simdata(nrows=1000, ncols=100, beta=NULL)
```

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

```r
# Extract simulated survival data
xs = simdata$xs         # matrix of predictors
y_ = simdata$y_         # vector of Gaussian (normal) outcomes
yb = simdata$yb         # vector of binomial outcomes
yt = simdata$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), quote=0 )
```

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

```r
# Check the rank of the design matrix, i.e. the degrees of freedom in the predictors
# using function from the Matrix package
Matrix::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  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

Because the values for lambda and gamma informed by CV are specifically chosen to give a best fit, model fit statistics for the CV informed model, when based upon the same data used to derive the model, will be biased. (Using the common terminology of machine learning, both the "training" and "validation" data inform the model fit.) To address this one can perform a CV on the CV derived estimates, that is a nested cross validation as argued for in Simon et al.. For this second layer of CV, there is no usage of information from the hold out data back to the model fit. (Using the common terminology of machine learning, each of the hold-out subsets of this outer layer of CV is treated as a "test" data set for calculation of model performance, and the results combined across these multiple hold out "test" sets.) 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.glmnetr() function which first fits all models based upon all data and then performs the cross validation for calculation of concordances or R-squares, deviance ratios and linear calibration summaries.

```
set.seed(465783346)
nested.cox.fit = nested.glmnetr(xs, NULL, yt, event, family="cox",
                 dolasso=1, alpha=seq(0,1,0.2), dostep=1, dofull=1, steps_n=40, folds_n=10, track=1)
```

Note, in the derivation of the relaxed lasso model fits, individual coefficients may be unstable even when the model may be stable (see the "Lasso and Ridge, Model and Coefficient Stability" vignette) which elicits warning messages. We suppress these warnings here. The first term in the call to nested.glmnetr(), xs, is the design matrix for predictors. The second input term, here given the value 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, here yt, 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 yt. The forth 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. We recommend setting track=1 when running the program interactively. Depending on the size of the data set and the different machine learning models fit, run time can be long and it can be helpful to view the progress in calculations.

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 before proceeding to describe model performances. The data summary includes sample size, number of events, number of candidate model predictors, degress of freedom in these predictors as well as average deviance and some average minus 2 log likelihoods. 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. Hyperparamters considered for stepwise regression are degress of freedom (df) and p, the p-value for entry into the regression equation, as discussed by James et al.. Performance measures include deviance ratio, linear calibration coefficients and measures of agreement, here for the Cox model framework concordance. Additionally there are the deviance ratio and agreement values naively calculated on the whole data set.

```
# Summarize relaxed lasso model performance informed by cross validation
summary(nested.cox.fit, width=84)
```

```
##   Sample information including number of records, events, number of columns in
##   design (predictor, X) matrix, and df (rank) of design matrix:
##               family                    n               nevent
##                  cox                 1000                  698
##           xs.columns                xs.df      null.dev/nevent
##                  100                   94                12.43
## null.m2LogLik/nevent  sat.m2LogLik/nevent
##                12.43                    0
##
##  For LASSO, and Stepwise regression tuned by df and p, average (Ave) model
##  performance measures from the 10-fold (NESTED) Cross Validation are given together
##  with naive summaries calculated using all data without cross validation
##
##                       Ave DevRat Ave Slope Ave Concordance Ave Non Zero
## lasso                     0.2412    1.1202          0.8717         44.0
## lassoR                    0.2425    1.0316          0.8718         17.5
## lassoR0                   0.2400    0.9693          0.8728         16.1
## elastic net               0.2454    1.0698          0.8734         22.1
## ridge                     0.2316    1.1232          0.8666         99.0
## elastic net G0            0.2422    0.9715          0.8740         18.1
## elastic net G1            0.2412    1.1202          0.8717         44.0
##                       Naive DevRat Naive Concordance Non Zero
```

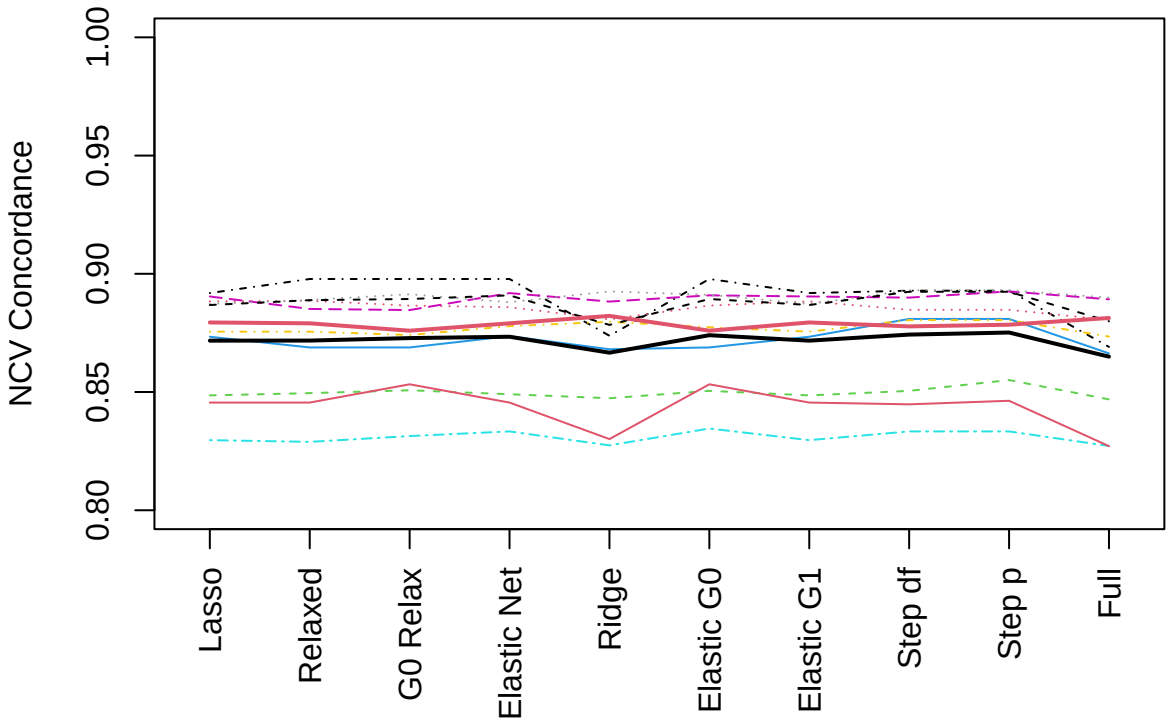
```
## lasso                        0.1696            0.8794       45
## lassoR                       0.1710            0.8791       20
## lassoR0                      0.1663            0.8759       14
## elastic net                  0.1710            0.8791       20
## ridge                        0.1718            0.8822       99
## elastic net G0               0.1663            0.8759       14
## elastic net G1               0.1696            0.8794       45
##
##                        Ave DevRat Ave Slope Ave Concordance Ave Non Zero
## Stepwise df tuned          0.2496    0.9941          0.8743         14.7
## Stepwise p tuned           0.2522    1.0010          0.8752         14.8
## Full regression            0.2251    0.0000          0.8650         93.0
##                        Naive DevRat Naive Concordance Non Zero
## Stepwise df tuned            0.1700            0.8778       14
## Stepwise p tuned             0.1711            0.8785       15
## Full regression              0.1797            0.8813       93
```

Here we see, not unexpectedly, that the concordances estimated from the nested CV are smaller than the concordances naively calculated using original data set. Depending on the data the nested CV and naive agreement measures, here concordance, can be very similar or disparate. Possibly surprisingly the deviance ratios are larger for the (nested) cross validation than naively calculated using all data. This flip in direction has not to do with the strength of association being stronger in the hold out data, but from the hold out data set being smaller and how this impacts average risk set size in the partial likelihood of the "cox" model. Such a flip in direction will generally not be the case for "binomial" and "gaussian" data. Despite this flip, the value of the cross validation estimated deviance ratios for comparing the machine learning models remains.

From this output we also see the number of non-zero coefficients in the different models, reflecting model complexity at least for the lasso model, along with the linear calibration coefficients obtained by regressing the outcome on the predicteds, i.e. the XBeta or the log(hazard ratio). (Many machine learning fitting routines use as predicteds the hazard ratio for "cox" model generalizations or the probability for "binomial" data. We give predicteds as the XBeta term which applies to the different data types. For the "binomial" case this is log(P/(1-P)) where P is the predicted probability.)

Model performance measures from the nested cross validation (NCV) can also be visualized with a plot which shows the calculated performances for the individual folds of the cross validation.

```
# Summarize relaxed lasso Cox model performance informed by cross validation
plot(nested.cox.fit, type="agree", ylim=c(0.8,1.0))
```



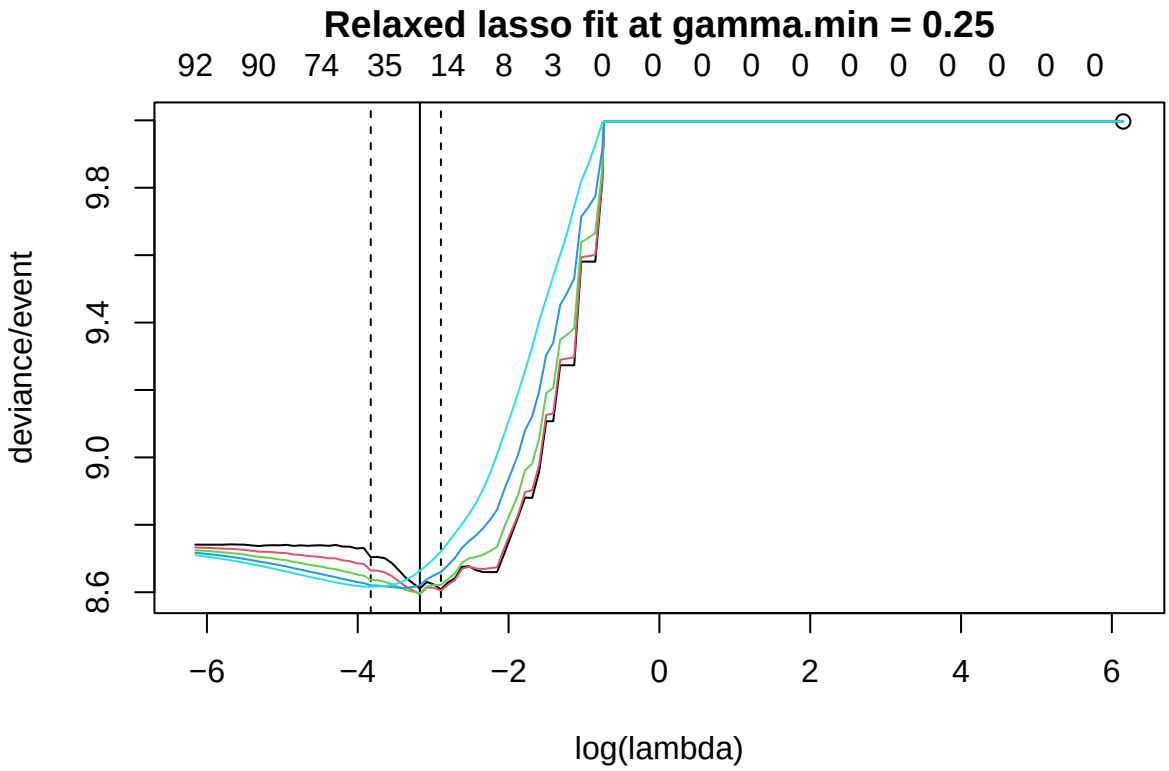
The performance measure estimates, here of concordance, from the individual (outer) cross validation for each fold are depicted by thin lines of different colors and styles, while the composite value (average) from all folds is depicted by a thicker black line, and the performance estimates naively calculated on all data, the same as the data used for model derivation, are depicted in a thicker red line. Here we see that the performance measures for the different models are quite variable across the folds, yet highly correlated with each other. Also, as expected the concordance for the model derived uisng the the full data set, naively calculated on the same data (i.e. the full data set) as used in model derivation, are larger than the average of the concordances from the different folds. Plots can also be constructed using option "devrat" for deviance ratios, "intcal" for linear calibration intercept coefficients and "lincal" for linear calibration slope coefficients.

The CV informed relaxed lasso model fit

As mentioned, the nested.glmnetr() function also derives the models using all data and stores these for further examination and usage. The choice of lambda and gamma hyperparameters for the relaxed lasso model is based upon a search across two dimensions for the pair that maximizes the likelihood, or similarly minimizes the deviances, as informed by a cross validation. The next plot depicts the deviances across these two dimensions for the full data set.

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

```
##  Relaxed lasso minimizing CV average deviance (maximizing log likelihood)
##      at gamma.min = 0.25, log(lambda) = -3.176, df = 20, deviance = 8.5943
##    fully relaxed (gamma=0) at log(lambda) = -2.897, df = 14, deviance = 8.6087
##    fully penalized (gamma=1) at log(lambda) = -3.827, df = 45, deviance = 8.6163
```

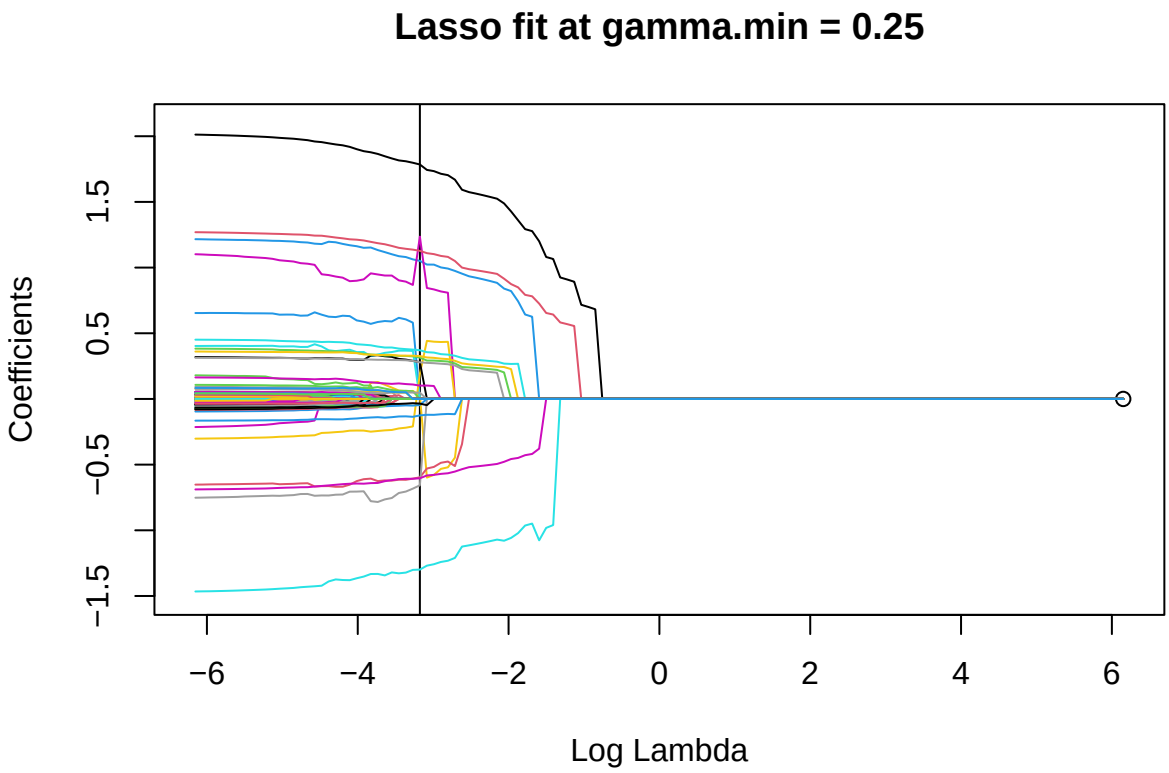


This figure has multiple lines, depicting deviance as a function of lambda for different gamma values. Whereas there is no legend here for gamma, when non-zero coefficients start to enter the model as the penalty is reduced, here shown at the right, deviances tend to be smaller for gamma = 0, greater for gamma = 1 and for small gamma values the opposite. The minimizing lambda and gamma pair is indicated by the solid vertical line, here about log(lambda) = -3.18. The minimizing lambda can be read from the horizontal axis. Because the different lines depicting deviances for the different values of gamma can be nearly overlapping, the minimizing gamma is described in the title, here 0.25. From this figure we also see that at log(lambda)=-3.18 the deviance is hardly distinguishable for gamma ranging from 0 to 0.5. Also 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 log(lambda) = -2.9, and highlighted by the right most back dashed vertical line. The fully penalized lasso model fits (gamma=1) shown in a cyan line achieves a minimal deviance at about log(lambda) = -3.83, and highlighted by the left most black dashed vertical line.

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, type="coef")
```

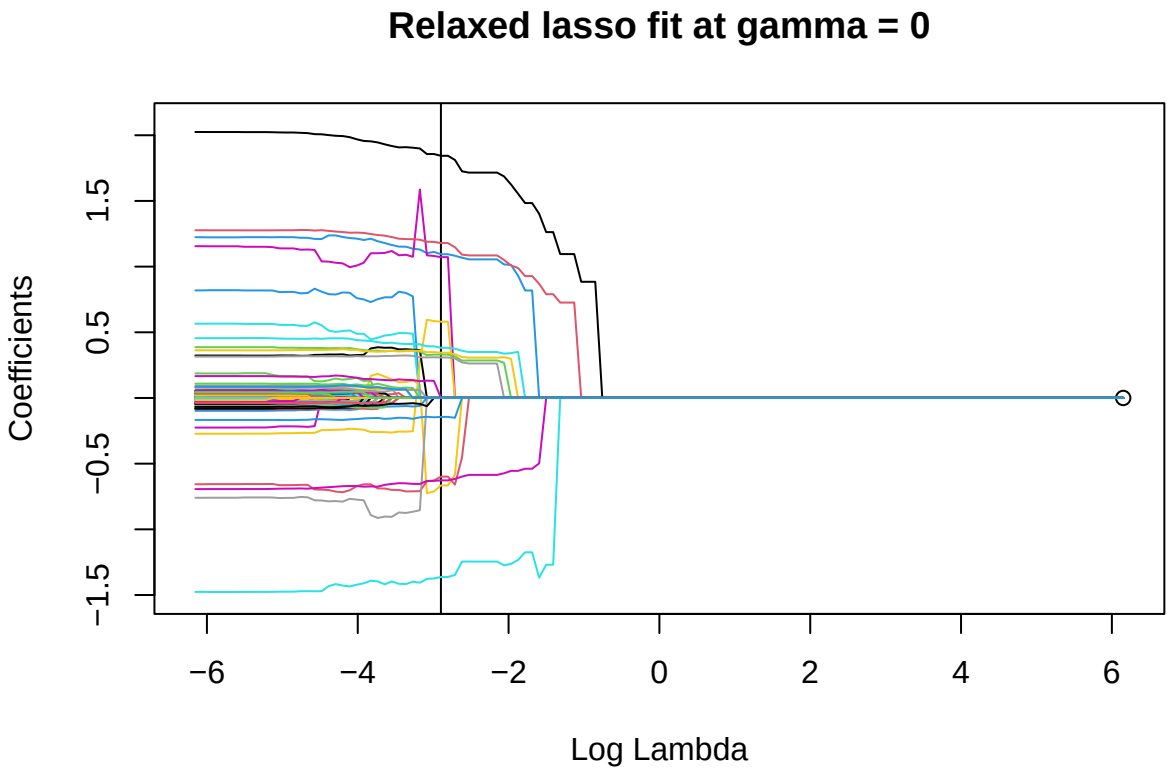
```
##   For lasso fit at gamma.min = 0.25,
##   log(lambda.min) = -3.1761
```



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 title, 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.

```r
# Plot fully relaxed coefficients informed by a cross validation
plot(nested.cox.fit, type="coef", gam=0)
```

```
##   For relaxed lasso fit at gamma = 0,
##   log(lambda.min) = -2.897
```



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, , to be substantially less than the 20 from the relaxed lasso fit and the 45 from the fully penalized lasso fit.

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

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

```
##   For the lasso model fits:
##   The minimum deviance is obtained at gamma = 0.25 and lambda = 0.041747
##   with df (number of non-zero terms) = 20, average deviance = 8.594314 and beta =
##          X4          X5          X7         X10         X14         X15
##  1.05097774 -1.29908396  0.02840708 -0.59677316  1.23560956  0.19246700
##         X16         X17         X18         X19         X20         X21
## -0.65735371  0.27975394  1.12772785  0.31495949 -0.12534339  0.37074447
##         X22         X23         X24         X25         X38         X60
```

```
## -0.60396706  0.32637480  0.27892897  1.78481244  0.10119098 -0.04634209
##         X88         X97
##  0.04739903 -0.03556323
##
##   The fully relaxed (gamma=0) minimum deviance is obtained at lambda = 0.055187
##   with df (number of non-zero terms) = 14, average deviance = 8.608703 and beta =
##         X4         X5         X7        X10        X14        X15        X18
##  1.0941317 -1.3627708 -0.6659704 -0.5988737  1.0720536  0.5804265  1.1792266
##        X19        X20        X21        X22        X23        X24        X25
##  0.3302200 -0.1444542  0.3820768 -0.6270976  0.3448214  0.3085739  1.8435689
##
##   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"  "X15"  "X38"  "X97"  "X16"  "X17"  "X60"  "X88"
## [21] "X12"  "X13"  "X43"  "X71"  "X100" "X34"  "X32"  "X50"  "X58"  "X41"
## [31] "X49"  "X64"  "X84"  "X91"  "X98"  "X39"  "X40"  "X66"  "X73"  "X74"
## [41] "X11"  "X61"  "X69"  "X70"  "X96"
```

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 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 predictor matrix xs_new, then the predicteds xs_new*beta are generated. When used to extract the model coefficients, the predict() function provides an output object in vector form (actually a list with two vectors) and so can easily be used for further calculations. By default the predict() 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, i.e. select the lambda that minimizes the CV deviance 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 in this function.

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

```
##    gamma.min = 0.25   lambda.min =  0.04174656   df = 20   deviance = 8.5943
```

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

```
##          X4          X5          X7         X10         X14         X15
##  1.05097774 -1.29908396  0.02840708 -0.59677316  1.23560956  0.19246700
##         X16         X17         X18         X19         X20         X21
```

```
## -0.65735371  0.27975394  1.12772785  0.31495949 -0.12534339  0.37074447
##         X22         X23         X24         X25         X38         X60
## -0.60396706  0.32637480  0.27892897  1.78481244  0.10119098 -0.04634209
##         X88         X97
##  0.04739903 -0.03556323
```

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

```
##          X1          X2          X3          X4          X5          X6
##  0.00000000  0.00000000  0.00000000  1.05097774 -1.29908396  0.00000000
##          X7          X8          X9         X10         X11         X12
##  0.02840708  0.00000000  0.00000000 -0.59677316  0.00000000  0.00000000
##         X13         X14         X15         X16         X17         X18
##  0.00000000  1.23560956  0.19246700 -0.65735371  0.27975394  1.12772785
##         X19         X20         X21         X22         X23         X24
##  0.31495949 -0.12534339  0.37074447 -0.60396706  0.32637480  0.27892897
##         X25         X26         X27         X28         X29         X30
##  1.78481244  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000
##         X31         X32         X33         X34         X35         X36
##  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000
##         X37         X38         X39         X40         X41         X42
##  0.00000000  0.10119098  0.00000000  0.00000000  0.00000000  0.00000000
##         X43         X44         X45         X46         X47         X48
##  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000
##         X49         X50         X51         X52         X53         X54
##  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000
##         X55         X56         X57         X58         X59         X60
##  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000 -0.04634209
##         X61         X62         X63         X64         X65         X66
##  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000
##         X67         X68         X69         X70         X71         X72
##  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000
##         X73         X74         X75         X76         X77         X78
##  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000
##         X79         X80         X81         X82         X83         X84
##  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000
##         X85         X86         X87         X88         X89         X90
##  0.00000000  0.00000000  0.00000000  0.04739903  0.00000000  0.00000000
##         X91         X92         X93         X94         X95         X96
##  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000
##         X97         X98         X99        X100
## -0.03556323  0.00000000  0.00000000  0.00000000
```

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

```
##    gamma.min = 0.25   lambda.min =  0.04174656   deviance = 8.5943
```

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

```
##  [1] -0.1724661 -3.0182290  4.7886317  1.8148686  0.3221870  1.7624826
##  [7] -3.3888967  0.8177817  6.5877625  2.0153974  1.5641699 -1.2664253
## [13]  1.2666317  3.2261061 -0.1800637  1.0035691  1.3183962  3.8324837
## [19] -2.0216757  2.4381785
```

Nested cross validation (NCV) for multiple models

Here we evaluate multiple machine learning models, in particular the lasso, ridge, XGB, random forest and neural network models. For this example we perform an analysis for the generalizations of linear regression in contrast to the Cox model in the last example. The glmnetr.simdata() function used above actually creates an output object list containing xs for the predictor matrix, yt for time to event or censoring and an event indicator, as well as y_ for "gaussian" and yb for "binomial" data.

```
# Nested cross validation evaluating a machine learning model suite with guassian errors
# Use the same simulated data output object from above, that is from the call
# simdata=glmnetr.simdata(nrows=1000, ncols=100, beta=NULL)
#
# recall linear regression model data generated above with line
# y_ = simdata$y_                  # outcome vector with Gaussian (normal) errors
# Get the ML model fits
nested.gau.fit = nested.glmnetr(xs,NULL,y_,NULL,family="gaussian",
      dolasso=1, doxgb=list(nrounds=250), dorf=1, doorf=1, doann=list(bestof=10),
      folds_n=10, seed=219301029, track=1)
```

```
# Summarize the results
summary(nested.gau.fit, width=84)
```

```
##   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/n
##   gaussian       1000        100         94       8.09
##
##  For LASSO, gradient boosting machine using XGBoost (XGB), Random Forest (RF),
##  Oblique Random Forest (ORF), Artificial Neural Networks (ANN), average (Ave) model
##  performance measures from the 10-fold (NESTED) Cross Validation are given together
##  with naive summaries calculated using all data without cross validation
##
##                     Ave DevRat Ave Int Ave Slope Ave R-square Ave Non Zero
## lasso                   0.8607 -0.0192    1.0184       0.8612         61.2
## lassoR                  0.8598 -0.0148    1.0138       0.8600         43.9
## lassoR0                 0.8570  0.0059    0.9899       0.8569         29.7
## ridge                   0.8548 -0.0433    1.0504       0.8574         99.0
##                     Naive DevRat Naive R-square Non Zero
## lasso                     0.8794         0.9380       66
## lassoR                    0.8794         0.9380       66
## lassoR0                   0.8763         0.9361       26
## ridge                     0.8806         0.9400       99
##
##                     Ave DevRat Ave Int Ave Slope Ave R-square Ave Non Zero
## XGB (not tuned)         0.6840 -0.0896    1.1174       0.6914          100
## XGB Tuned               0.8032 -0.1114    1.1054       0.8113          100
##                     Naive DevRat Naive R-square Non Zero
```

```
## XGB (not tuned)           0.9988         0.9988      100
## XGB Tuned                 0.9560         0.9585      100
##
##                      Ave DevRat Ave Int Ave Slope Ave R-square Ave Non Zero
## RF Simple               0.7117 -0.2305    1.2175        0.737           58
##                      Naive DevRat Naive R-square Non Zero
## RF Simple                  0.9285         0.9491       60
##
##                      Ave DevRat Ave Int Ave Slope Ave R-square Ave Non Zero
## ORF Simple              0.6674 -0.7946    1.7067       0.8096           33
##                      Naive DevRat Naive R-square Non Zero
## ORF Simple                 0.8867         0.9605       40
##
##                      Ave DevRat Ave Int Ave Slope Ave R-square Ave Non Zero
## ANN Uninformed          0.7083  0.0315    0.9722       0.7111          100
##                      Naive DevRat Naive R-square Non Zero
## ANN Uninformed             0.9599           0.96      100
```

Here we see a set of machine learning models evaluated together. 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, XGB, 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. random forest (and sometimes the XGB), the same seed is set using set.seed() before each model call facilitating replicability of results. The "Non Zero" column is the number of non zero regression coefficients for the lasso and ridge models, and the number of predictors given to the XGB model. For the random forest model "Non Zero" is the number of predictors randomly selected for possible splitting at each node, a tuned hyperparameter. For the ANN models "non Zero" again is the number of terms given to the ANN for training.

One can save the tabled information from a summary to a data frame using the *table* option as in

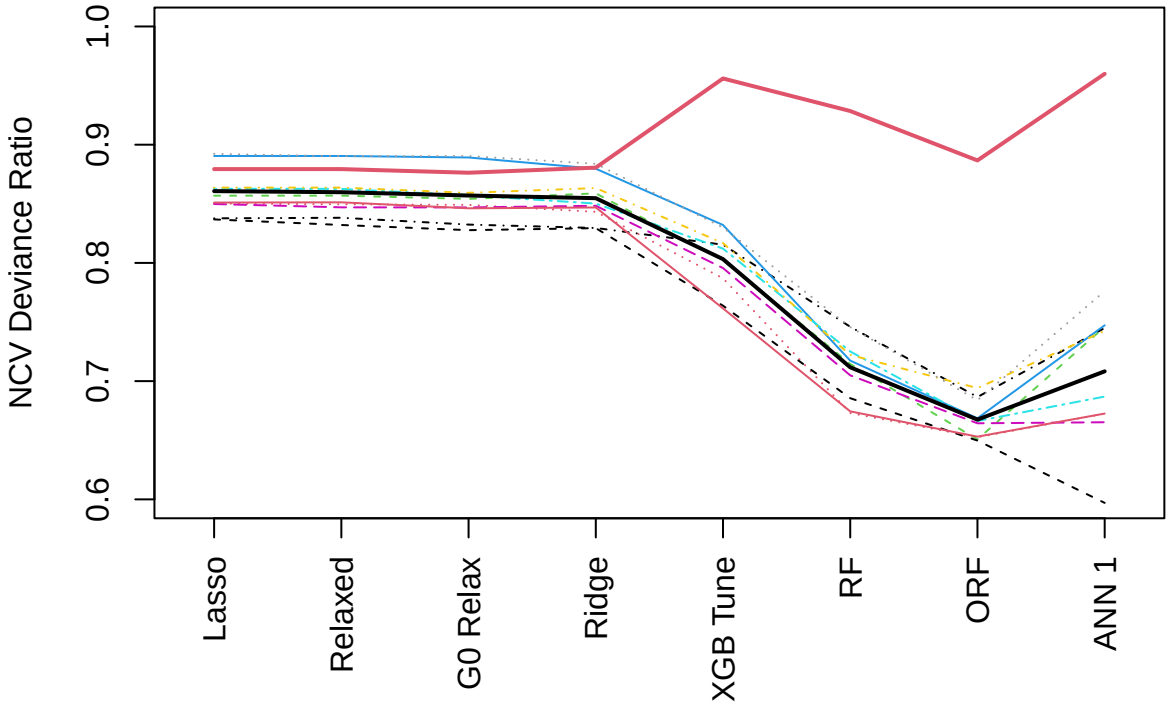
```
## put the data into a data frame
dframe = summary( nested.gau.fit , table = 0)
## use the roundperf() function from 'glmnetr' to round so long as it doesn't too
## obscure the original values
## print just the first 4 columns to get a more succinct table
roundperf(dframe, digits = 3)[,c(1:4)]
```

```
##                            Ave DevRat Ave Int Ave Slope Ave R-square
## lasso                           0.861  -0.019     1.018        0.861
## lassoR                          0.860  -0.015     1.014        0.860
## lassoR0                         0.857   0.006     0.990        0.857
## ridge                           0.855  -0.043     1.050        0.857
## XGB (not tuned)                 0.684  -0.090     1.117        0.691
## XGB Tuned                       0.803  -0.111     1.105        0.811
## RF Simple                       0.712  -0.231     1.217        0.737
## ORF Simple                      0.667  -0.795     1.707        0.810
## ANN Uninformed                  0.708   0.032     0.972        0.711
```

This may be helpful when wanting to further process the findings or incorporate into reports.

Model performance measures from the nested cross validation can be plotted individually as shown here for deviance ratio

```
# Summarize relaxed lasso model performance informed by cross validation
plot(nested.gau.fit, type="devrat", ylim=c(0.6,1))
```

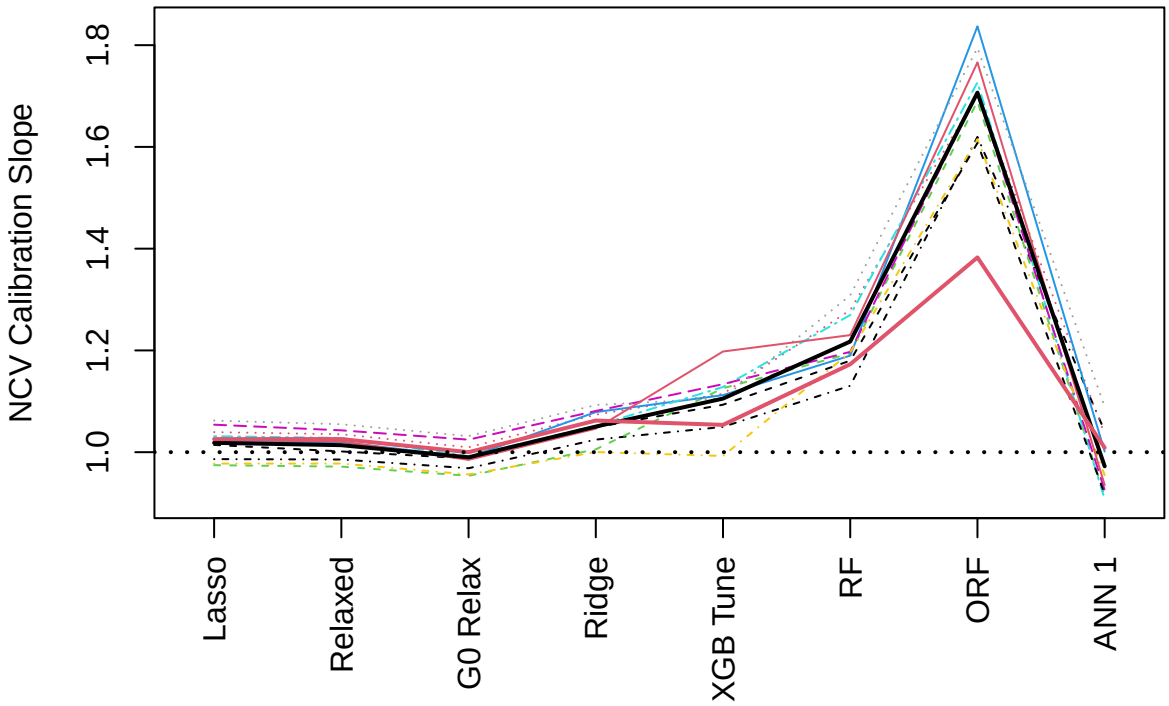


For "gaussian" data, i.e generalization of the linear regression model, the deviance ratio is the reduction Mean Square Error (MSE) achieved by the machine learning model relative to the MSE of the model based upon only the overall average.

While of less importance than deviance ratio one can (by using the options type="agree" and pow=2) also describe model performances in terms of R-square for "gaussian" data. Note, from Jensens' inequality we expect the R-squares for the different folds to be more biased than the R 's (correlation), and thus calculate the CV estimate for R-square as the (average R from the CV folds)-squared instead of average (R-squared from the CVs folds).

To understand how the models might be over or underestimating the strength of the association between the predictors and outcome one can regress the outcome on the model X*beta predicteds, i.e. to a linear calibration. Slope terms from this linear calibration can be plotted as in

```
# Summarize relaxed lasso model performance informed by cross validation
plot(nested.gau.fit, type="lincal")
```



Model and Cross Validation storage, and reuse

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.tuned.fit and cv.stepreg.fit. cv_glmnet_fit has a similar yet different format to that of cv.glmnet() by including further fit information. The XGB outputs are essentially direct outputs from 'xgboost' xgb.train(), but with added information regarding the seed or fold used in the fit as well as the parameters used in the search for the hyperparamaters using the 'mlr3mbo' package. The rf.tuned.fit object contains the output from rfsrc() in the object rf.tuned$rf_tuned along with 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 data sets like xx.devian.rep, xx.lincal.rep, xx.agree.rep for further processing by the summary() function or by the user. For example

```
# Manually calculate CV R_square for lasso models
corr.rep = nested.gau.fit$lasso.agree.rep
corr.rep
```

```
##           lasso    lassoR   lassoR0 elastic net     ridge elastic net G0
##  [1,] 0.9161852 0.9164505 0.9136802   0.9164505 0.9121208      0.9136802
##  [2,] 0.9229059 0.9224611 0.9217117   0.9224611 0.9205015      0.9217117
##  [3,] 0.9270295 0.9271735 0.9261634   0.9271735 0.9283240      0.9261633
##  [4,] 0.9438189 0.9438189 0.9430161   0.9438189 0.9404214      0.9430160
##  [5,] 0.9293502 0.9292241 0.9260119   0.9292241 0.9236872      0.9260118
##  [6,] 0.9236294 0.9214803 0.9208406   0.9214803 0.9244237      0.9208406
##  [7,] 0.9304886 0.9304886 0.9288058   0.9304886 0.9303647      0.9288058
```

```
##  [8,] 0.9469188 0.9453269 0.9446574   0.9453269 0.9441954      0.9446574
##  [9,] 0.9157996 0.9132125 0.9109753   0.9132125 0.9128563      0.9109752
## [10,] 0.9238633 0.9238359 0.9210755   0.9238359 0.9224755      0.9210754
##       elastic net G1
##  [1,]      0.9161852
##  [2,]      0.9229059
##  [3,]      0.9270295
##  [4,]      0.9438189
##  [5,]      0.9293502
##  [6,]      0.9236294
##  [7,]      0.9304886
##  [8,]      0.9469188
##  [9,]      0.9157996
## [10,]      0.9238633
```

```
avecorr = colMeans(corr.rep)
R_square = round( avecorr ^2 , digits = 4)
R_square
```

```
##          lasso         lassoR        lassoR0    elastic net          ridge
##         0.8612         0.8600         0.8569         0.8600         0.8574
## elastic net G0 elastic net G1
##         0.8569         0.8612
```

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

Further model assessment

A naive calibration

Further model assessment can be made based upon the predicteds from the predict() function. For example, one can model the outcomes based upon a spline for the X*Beta hats from the predicteds. This may help to understand potential nonlinearities in the model, but may also give inflated hazard ratios.

```
# Get predicteds 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
```

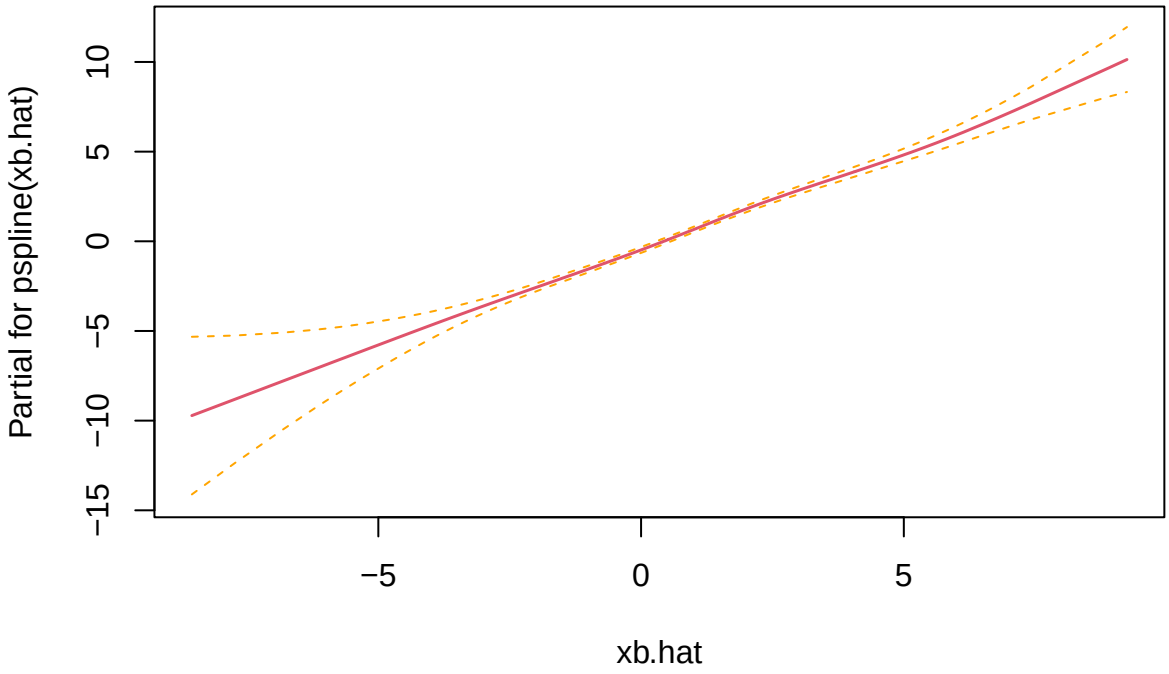
```
##     1%     5%    10%    25%    50%    75%    90%    95%    99%
## -5.367 -3.650 -2.761 -1.332  0.403  2.050  3.660  4.462  5.921
```

```
# Fit a spline to xb.hat uisng coxph, and plot
library(survival)
fit1 = coxph(Surv(yt, event) ~ pspline(xb.hat))
print(fit1)
```

```
## Call:
## coxph(formula = Surv(yt, event) ~ pspline(xb.hat))
##
```

```
##                              coef se(coef)      se2    Chisq   DF      p
## pspline(xb.hat), linear 1.07e+00 3.33e-02 3.33e-02 1.03e+03 1.00 <2e-16
## pspline(xb.hat), nonlin                              3.77e+00 3.04   0.29
##
## Iterations: 4 outer, 16 Newton-Raphson
##      Theta= 0.738
## Degrees of freedom for terms= 4
## Likelihood ratio test=1494  on 4.04 df, p=<2e-16
## n= 1000, number of events= 698
```

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



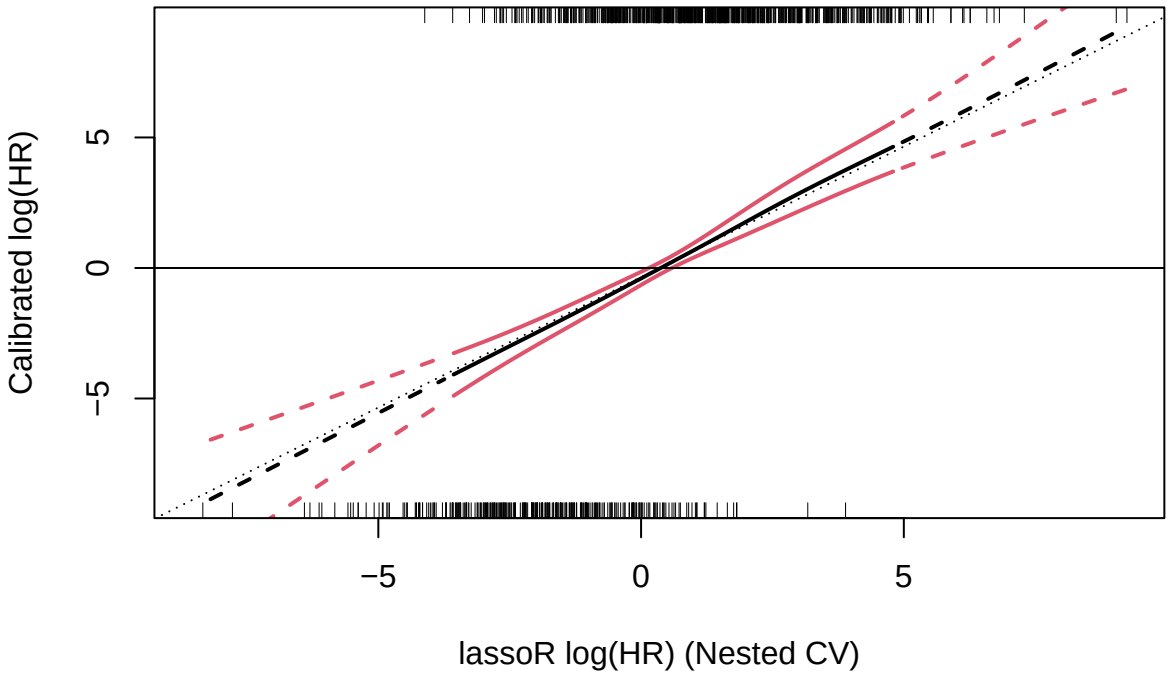
From this spline fit we see the log hazard ratio is roughly linear in the relaxed lasso predicteds. Still, a calibration based upon data not used in model derivation would be more meaningful.

Hold out data calibrations

The calplot() function generates calibration plots using spline fits for each of the hold out data sets, either form the outer loop of the nested cross validation or the out of bag sample partitions when using a bootstrap approach, for example based upon cross validation

```
calplot(nested.cox.fit,wbeta=3)
```

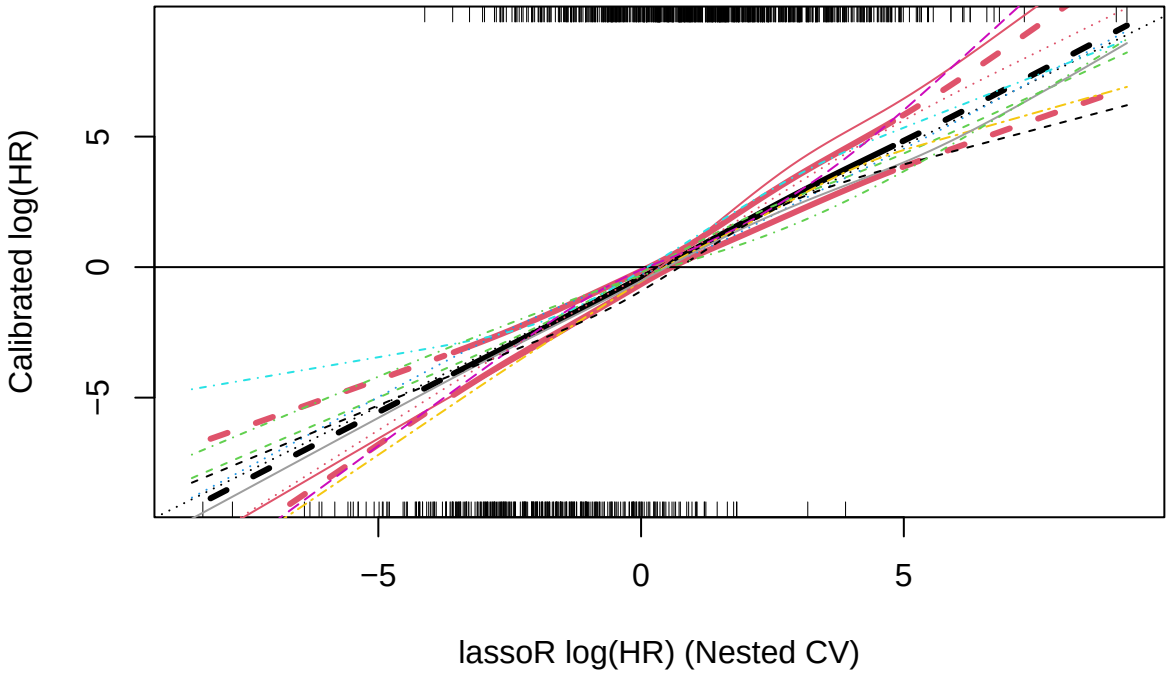
```
##   Range of X*Beta for calibration:
## -8.339908 9.246226
##   Range of calibrated naive confidence intervals:
## -11.62559 11.66253
```



Here the "overall" calibration and confidence interval are calculated using the values from the individual folds, taking the mean as usual and adjusting the usual SE by multiplying by sqrt(1+k/(k-1), i.e. the Nadeau-Bengio adjusted formula for SE (Nadeau et al.), and then calucalting mean +/- 2 standard error for the confidence intervals. The Nadeau-Bengio adjustment might not be derived for these types of statistics but might serve well as an approximation. One can also include the individual calibration curves from the outer cross validation loop with the option plotfold=1 as in

```
calplot(nested.cox.fit, wbeta=3, plotfold=1 )
```

```
##   Range of X*Beta for calibration:
## -8.339908 9.246226
##   Range of calibrated naive confidence intervals:
## -11.62559 11.66253
```



Calibration is discussed in more detail in the "Calibration of Machine Learning Models" vignette.

Some limitations of cross validation

Bengio et al. showed that the standard deviations from cross validation might not be accurate for estimation of the actual standard errors. Bates et al. have shown that the cross validation (CV) estimates and standard errors may be biased, and should thus be viewed with caution. These considerations regard the cross validation for evaluation of model performance, here the outer loop, and are agnostic to whether or not the models use any cross validation for the derivation of the models, here the inner loop. In particular the CV estimates model performance may more closely estimate the expectation over multiple samples than the performance of the model based upon the observed data. In this manner the CV estimates may be reasonable estimates for the model fitting process, but may be biased for a particular model fit. Further, with their bias, the naive standard errors applied to CV folds, might be associated with mis-coverage three times the nominal mis-coverage. A resolution of these biases in estimation and standard error proposed by Bates at al. involves two more iteration loops lengthening computation time by a factor of about 10 when using 10-fold cross validation and possibly by a factor of 100 if one is to replicate the whole process multiple times to get more accurate estimates, and is not derived for all model performance measures. The Bates approach may be practical for machine learning models that run more quickly like possibly lasso and random

forests but impractical for models that can take much longer like tuned gradient boosting machines. The Bates approach is not implemented here.

Model comparisons with simulated and observed data

The examples above compare models using simulated data. Simulated data can often be constructed to show one or the other model to do perform the best. For the examples above the lasso models and those informed by the relaxed lasso performed better. When comparing models using observed medical data we have generally seen the lasso models to be consistently among the better performing models, but the differences in performances not as large and consistent as in these examples. Notably, when using observed data, the oblique random forest (ORF) often performed slightly (but not statistically significantly) better than the other models. By its construction we would expect the ORF to fit better when there are stronger correlations or interactions between predictors/features. In our test runs on observed data we often did have meaningful correlations between the predictors/features. Because different models may perform better with different types of data it may be informative to compare model performances for multiple data sets of a particular data type, when available, not only a single data set of immediate interest.

Model replicability and model comparisons

To facilitate reproducible results the nested.glmnetr() function stores the seeds used to generate the pseudo random samples used for assigning observations to folds in the outer loop, as well as the fold ids themselves, for all models. Additionally, the program stores the seeds when generating the folds in the inner loop for lasso, XGB, neural network and stepwise regressions as well as the seeds used when generating the bootstrap samples for the random forest models. Using the seeds from a prior call to nested.glmnetr(), one can reproduce the results from this earlier call. Because the seeds are saved for sample partitioning in both the outer cross validation or the bootstrap method along with the folds for inner loop model derivations, results can be reproduced even when rerunning an analysis for a single model. If we did not save and manage the seeds the user might have to run all models included in an earlier run to verify or inspect a single model fit. If the seed option is unspecified in the call to nested.glmnetr(), one should be cautious when using set.seed() in one's own code as this too will effect the pseudo randomness used in the ML calculations, which could unwittingly yield identical results when pseudo independent runs are intended.

Using the same folds, i.e. data splits, for the different models controls for some of the variability in the model performance estimates due to the randomness in the choice of folds, which should reduce variability in the differences in performance estimates between different models. nested.glmnetr() stores the model performance measures from each iteration of the outer loop allowing calculation of means and standard deviations (SD) for each model performance measure, as well as differences paired on each data split in the outer loop. Because of dependencies between the different model fits from a CV loop, there are concerns about the accuracy of the SDs (Bengio et al. and Bates et al.). Here, following the intuition of Nadeau and Bengio, we use the correction to the naive SE = SD/sqrt(k) of SE/sqrt(1/k+1/(k-1)).

A simple comparison of model performances based upon deviance ratio can be done as in the example

```
# compare fractional reductions in Mean Square Error between different models
nested.compare(nested.gau.fit, type="devrat")
```

```
##
##  Model performance comparisons in terms of ** Deviance Ratio **
##
##  p_NB is p by Nadeau-Bengio, p_n is naive p
##
```

```
##
##  Comparison                        devrat     lower 95%   upper 95%    p_NB
##  lasso relaxed - lasso            -0.0010  ( -0.0027   , 0.0008    )  0.2460
##  lasso relaxed - lasso relaxed G0  0.0029  (  0.0005   , 0.0054    )  0.0240
##  lasso relaxed - ridge             0.0049  ( -0.0003   , 0.0100    )  0.0607
##  lasso - lasso relaxed G0          0.0039  (  0.0014   , 0.0065    )  0.0071
##  p_n
##  0.1049
##  0.0034
##  0.0125
##  0.0007
##
##  Comparison                          devrat    lower 95%   upper 95%    p_NB
##  XGBoost (tuned) - XGBoost (simple) 0.1199  ( 0.076     , 0.1638    )  2e-04
##  p_n
##  0
##
##  Comparison           devrat    lower 95%   upper 95%    p_NB  p_n
##  step (df) - step (p) 3e-04   ( -0.0037   , 0.0044    )  0.861 0.7994
##
##  Comparison                   devrat     lower 95%   upper 95%    p_NB   p_n
##  lasso relaxed G0 - step (p) -0.0065  ( -0.0131   , 0         )  0.0509 0.0097
##
##  Comparison                   devrat    lower 95%   upper 95%    p_NB p_n
##  lasso relaxed - XGB (tuned) 0.0566  ( 0.0389    , 0.0742    )  0    0
##
##  Comparison                     devrat    lower 95%   upper 95%    p_NB p_n
##  lasso relaxed - Random Forest 0.1473  ( 0.1214    , 0.1733    )  0    0
##
##  Comparison                              devrat    lower 95%   upper 95%    p_NB
##  lasso relaxed - Oblique Random Forest 0.1912  ( 0.1702    , 0.2123    )  0
##  p_n
##  0
##
##  Comparison          devrat    lower 95%   upper 95%    p_NB p_n
##  lasso relaxed - ANN 0.1531  ( 0.1076    , 0.1987    )  0    0
##
##  Comparison           devrat    lower 95%   upper 95%    p_NB p_n
##  XGBoost (tuned) - RF 0.0908  ( 0.076     , 0.1055    )  0    0
##
##  Comparison            devrat    lower 95%   upper 95%    p_NB p_n
##  XGBoost (tuned) - ORF 0.1347  ( 0.1168    , 0.1526    )  0    0
##
##  Comparison            devrat    lower 95%   upper 95%    p_NB  p_n
##  XGBoost (tuned) - ANN 0.0966  ( 0.0585    , 0.1346    )  3e-04 0
##
##  Comparison devrat    lower 95%   upper 95%    p_NB  p_n
##  RF - ORF   0.0439  ( 0.0263    , 0.0615    )  3e-04 0
##
##  Comparison devrat    lower 95%   upper 95%    p_NB   p_n
##  RF - ANN   0.0058  ( -0.0348   , 0.0464    )  0.7546 0.6507
##
##  Comparison devrat     lower 95%   upper 95%    p_NB   p_n
##  ORF - ANN  -0.0381  ( -0.0858   , 0.0096    )  0.1043 0.0276
```

As the outer CV loop performance calculations are saved in the nested.glmntr() output object (with names like lasso.agree.rep, lasso.lincal.rep, xgb.agree.rep, xgb.lincal.rep, etc.) comparisons not provided by nested.compare() can be calculated from these stored CV data.

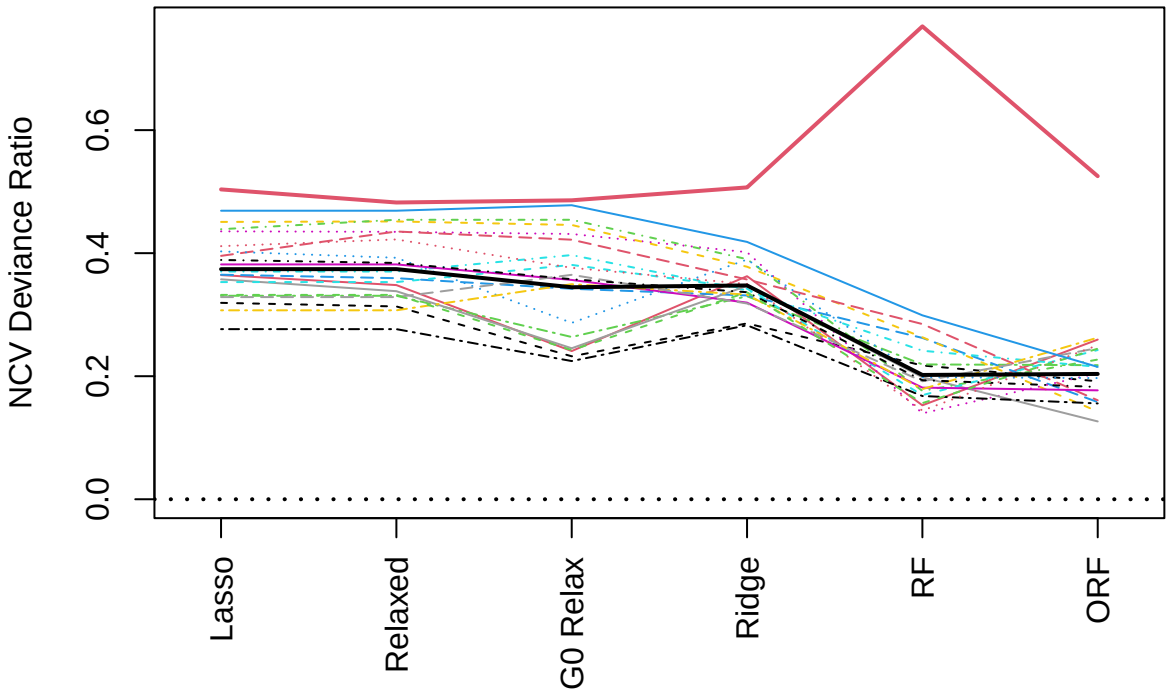
Bootstrap assessment of model performance

If one is uncertain about using cross validation for assessing model performance, one can implement bootstrap resampling, where the properties may be better understood. Bootstrap resampling might better mimic the randomness of model selection in the original model than the (outer loop of nested) cross-validation resampling and might provide more accurate performance estimates. When taking bootstrap resamples for machine learning refits we (primarily) evaluate performance on the hold-out data and not the bootstrap (train) resample (itself), though this approach is not universal (Riley et al.). While we have not systematically evaluated this our impression is that for the numerical assessments of model performance the differences between the bootstrap and cross-validation results were not especially large. However, the out-of-bag calibration plots can noticeably differ between the bootstrap and cross validation approaches. Depending on the models being fit and the data analyzed the run times can be very long for the bootstrap approach. For some analyses evaluating lasso, gradient boosting machines, artificial neural networks and stepwise models we have had runs for 10-fold nested cross-validation taking days. A bootstrap study with 50 or 100 resamplings could then take weeks. One approach might be to first evaluate performances using the cross-validation and then re-evaluate any unclear model performances using the bootstrap. Another approach would be to evaluate initially using the bootstrap with a small number of resamplings, e.g. 10 like sometimes used for cross-validation, trading stability for accuracy. Again one could selectively re-evaluate models of interest with a larger number of bootstrap resamplings. Bootstrap sampling can be implemented by specifying the bootstrap option for the number of bootstrap samples to be used, e.g. as in

```
nested.bin.boot.fit = nested.glmnetr(xs,NULL,yb,NULL,family="binomial",
      dolasso=1, dorf=1,
      folds_n=10, seed=219301029, bootstrap=20, track=1)
```

where models are fit for each of 20 re-samples from the original data. For each re-sample models are evaluated using the respective out-of-bag (OOB) data, i.e. the original data not included in the re-sample, and these are used to derive overall model performances.

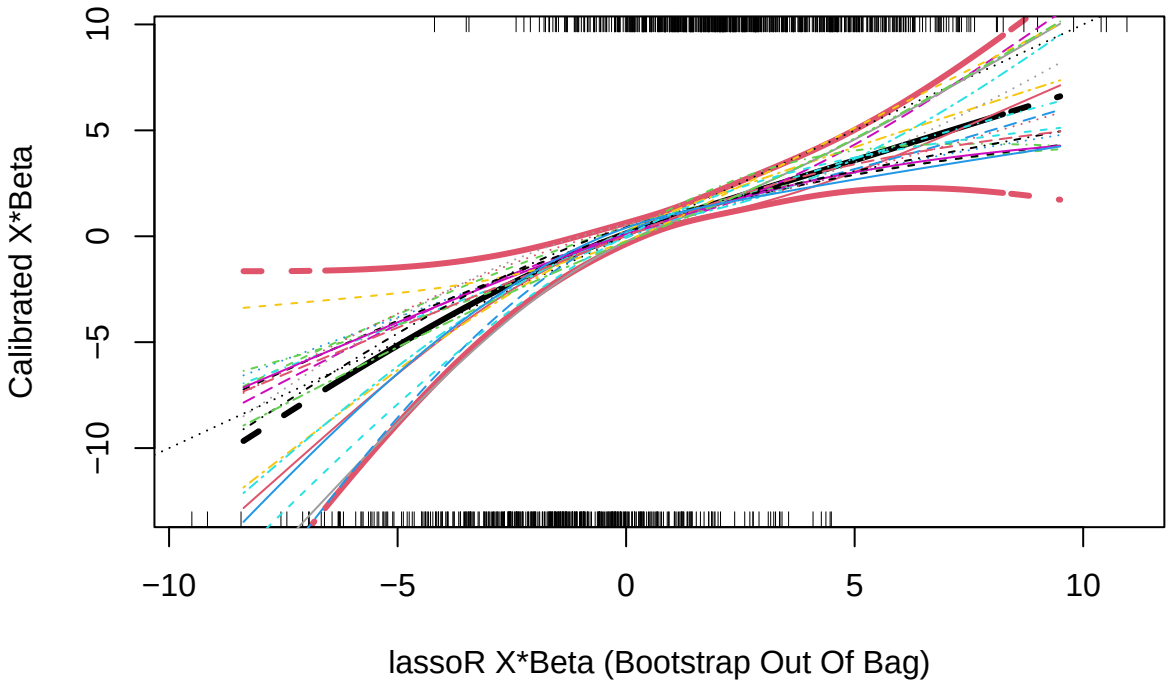
```
plot(nested.bin.boot.fit)
```



Calibration curves can be generated like with the cross validation model evaluations, as in

```
calplot(nested.bin.boot.fit, wbeta=3, plotfold=1)
```

```
##   Range of X*Beta for calibration:
## -9.501871 10.95765
##   Range of calibrated naive confidence intervals:
## -17.68289 11.49037
```



Here, similar to the cross-validation calibration plots, calibration curves are fit using splines for each bootstrap sample based upon the upon the XBeta's (predicteds) for the out-of-bag sample based upon the re-sample fitted model. These curves are then averaged to get an overall calibration curve. Nominal 95% confidence intervals are described based upon the overall curve and the standard deviation of the individual curves at each value of XBeta, though again the coverage of these nominal confidence intervals is in question. See the calibration vignette for more on calibration curves based upon bootstrap samples.

References

Bates S, Hastie T, Tibshirani R, "Cross-validation: what does it estimate and how well does it do it?", 2022, https://arxiv.org/abs/2104.00673 .

Bengio Y & Grandvalet Y, "No Unbiased Estimator of the Variance of K-Fold Cross-Validation", Journal of Machine Learning Research 5, 2004, 1089–1105, https://www.jmlr.org/papers/volume5/grandvalet04a/grandvalet04a.pdf .

James G, Witten D, Hastie T, Tibshirani R, "An Introduction to Statistical Learning with applications in R, 2nd ed.", Springer, New York, 2021.

Nadeau C, Bengio Y, "Inference for the Generalization Error", Machine Learning, 52, 239–281 (2003).

Riley RD, Pate A, Dhiman P, Archer L, Martin GP, Collins GS, "Clinical prediction models and the multiverse of madness", BMC Med 21, 502 (2023). https://doi.org/10.1186/s12916-023-03212-y

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