Calibration of Machine Learning Models in glmnetr

Walter K. Kremers, Mayo Clinic, Rochester MN

28 December 2024

The Package

The "An Overview of glmnetr" vignette shows how to run the main package function nested.glmnetr() and how to summarize model performances. If one identifies a well performing model according to the metrics in this summary, e.g. concordance, correlation, deviance ratio, linear calibration, one may want to do further evaluation in terms of calibration. The strongest calibration and validation will involve calibration with new, independent datasets. Frequently one will not have immediate access to such new data sets, or one may want first to do an internal validation before subjecting a model to an external validation. Here we consider an internal validation approach using cross validation or bootstrap re-sampling, similar to how we numerically assessed model performance.

An Example Analysis

To explore calibration we first consider the nested.glmnetr() call from the "An Overview of glmnetr" vignette which fit machine learning models to survival data with family="cox", i.e.

```
set.seed(465783345)
nested.cox.fit = nested.glmnetr(xs, NULL, yt, event, family="cox",
                                dolasso=1, dostep=1, steps_n=40, folds_n=10, track=1)
```

Linear Calibration

Using either print() or summary() on the output object nested.cox.fit one gets, amongst other information, summaries for the linear calibration slopes and intercepts as in

```
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               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 min                 0.2452    1.0702          0.8730         48.0
## LASSO minR                0.2470    1.0083          0.8744         21.0
## LASSO minR.G0             0.2435    0.9451          0.8733         16.8
## Ridge                     0.2256    1.2887          0.8660         99.0
##                       Naive DevRat Naive Concordance Non Zero
## LASSO min                   0.1696            0.8794       42
## LASSO minR                  0.1710            0.8791       20
## LASSO minR.G0               0.1663            0.8759       13
## Ridge                       0.1718            0.8822       99
##
##                       Ave DevRat Ave Slope Ave Concordance Ave Non Zero
## Stepwise df tuned         0.2541    0.9741          0.8776         14.7
## Stepwise p tuned          0.2549    0.9775          0.8786         15.0
##                       Naive DevRat Naive Concordance Non Zero
## Stepwise df tuned           0.1711            0.8785       15
## Stepwise p tuned            0.1711            0.8785       15
```

Here we see that for many of the models the linear calibration slope term is near 1, the ideal for perfect calibration. For the Cox model any intercept term can be absorbed into the baseline survival function and there is no pertinent intercept term for calibration.

A First Visual

An initial calibration consideration was made in the overview vignette by regressing observed outcomes on the predicteds from the final model based upon the relaxed lasso. This regression was made using splines, in particular the pspline() function from within a coxph() call, as in

```
# 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)
# Fit a spline to xb.hat uisng coxph, and plot
#library( survival )                  ## load survival package for Cox model fits
fit1 = coxph(Surv(yt, event) ~ pspline(xb.hat, df=3))
```

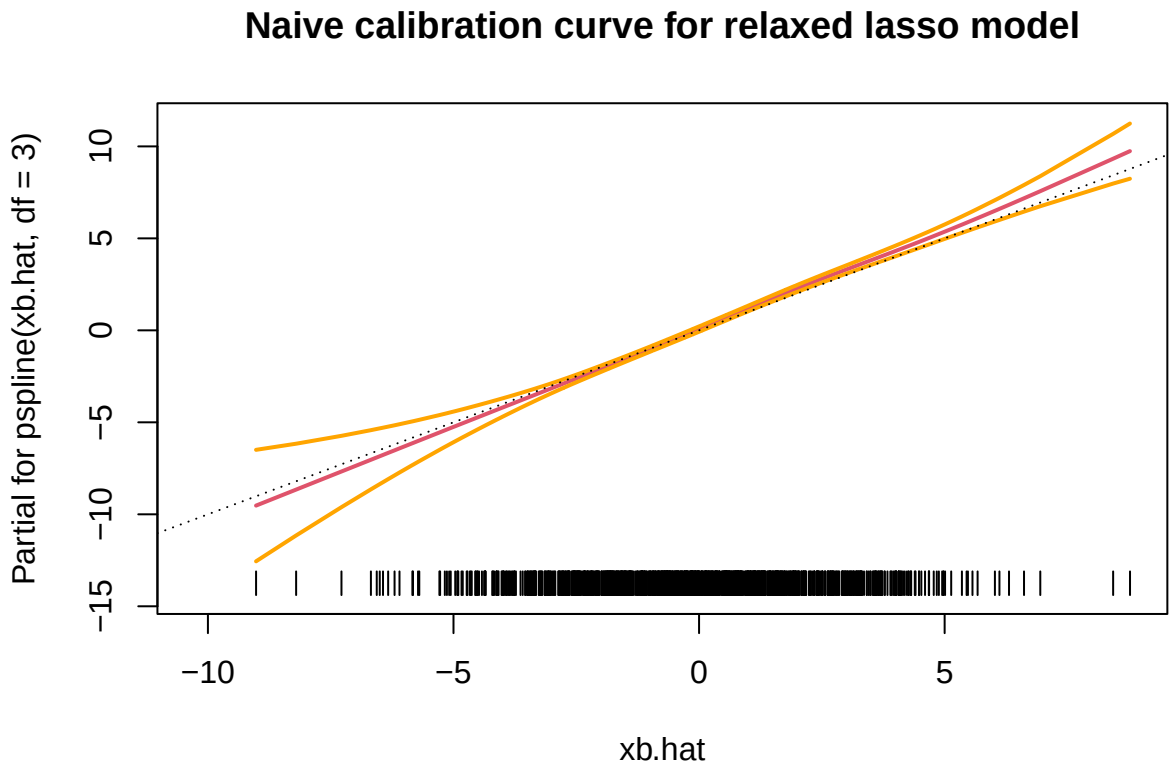
```
summary(fit1)
```

```
## Call:
## coxph(formula = Surv(yt, event) ~ pspline(xb.hat, df = 3))
##
##   n= 1000, number of events= 698
##
##                            coef  se(coef) se2     Chisq   DF   p
## pspline(xb.hat, df = 3),  1.073 0.03333  0.03333 1035.62 1.00 3.3e-227
## pspline(xb.hat, df = 3),                           2.17 2.03  3.4e-01
##
##              exp(coef) exp(-coef) lower .95 upper .95
## ps(xb.hat)3  1.068e+01  9.361e-02 1.642e+00 6.949e+01
## ps(xb.hat)4  1.141e+02  8.763e-03 4.179e+00 3.117e+03
```

```
## ps(xb.hat)5  1.217e+03  8.214e-04 1.710e+01 8.667e+04
## ps(xb.hat)6  1.244e+04  8.040e-05 1.183e+02 1.307e+06
## ps(xb.hat)7  1.324e+05  7.552e-06 1.269e+03 1.382e+07
## ps(xb.hat)8  1.714e+06  5.835e-07 1.661e+04 1.768e+08
## ps(xb.hat)9  1.443e+07  6.932e-08 1.380e+05 1.508e+09
## ps(xb.hat)10 1.719e+08  5.818e-09 1.575e+06 1.875e+10
## ps(xb.hat)11 2.481e+09  4.030e-10 1.910e+07 3.222e+11
## ps(xb.hat)12 3.678e+10  2.719e-11 1.773e+08 7.630e+12
##
## Iterations: 4 outer, 14 Newton-Raphson
##      Theta= 0.7945968
## Degrees of freedom for terms= 3
## Concordance= 0.879  (se = 0.005 )
## Likelihood ratio test= 1492  on 3.03 df,   p=<2e-16
```

followed by plotting with

```
termplot(fit1,term=1,se=TRUE, rug=TRUE, lwd.term=2, lwd.se=2, lty.se=1) # , col.term=1, col.se=2
abline(a=0,b=1,lty=3)
title ("Naive calibration curve for relaxed lasso model")
```



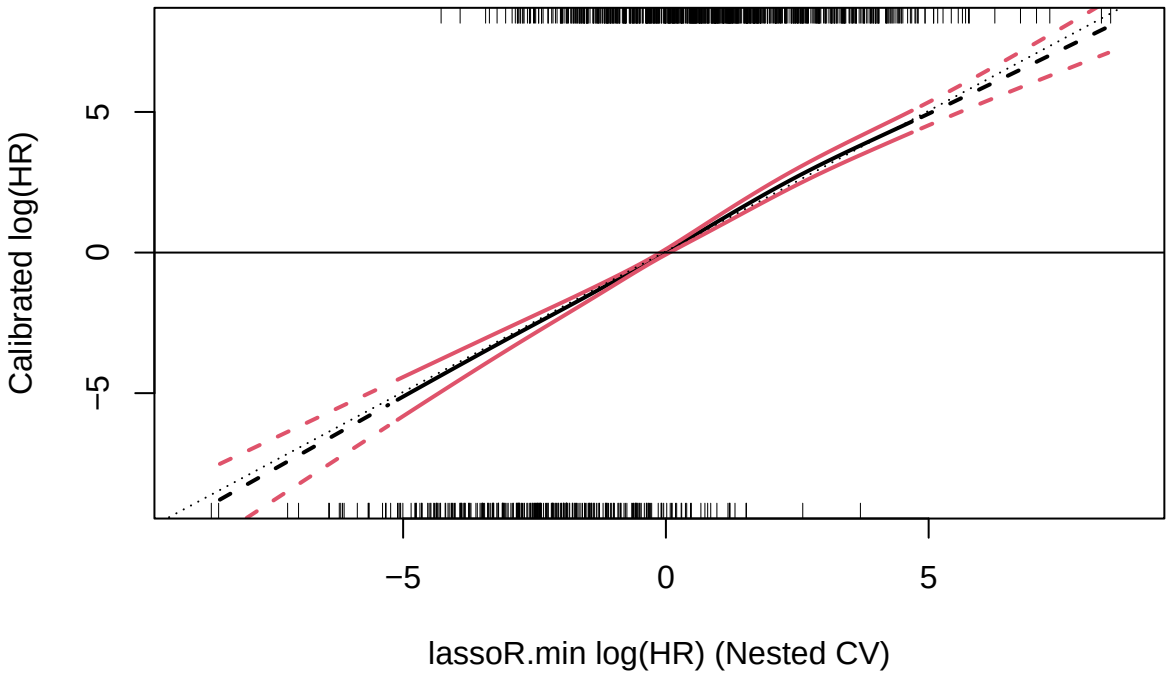
The spline fits may help to understand potential nonlinearities in the model. Here we see, a clibration line which is not far from linear. Still, as noted in the "An Overview of glmnetr" vignette, because the same data are used for model evaluation as well as model derivation, it is hard to put much confidence in such a calibration plot because of potential bias which may suggest a better fit than can be expected for new data.

Calibration Using Spline Fits and Resampling

For each of the models fit, nested.glmnetr() saves the X* Beta's from the final model. The nested.glmentr() function also calculates the X* Betas's for the hold out data for each partitioning, i.e. each hold out fold of the outer loop of nested cross validation or the out-of-bag items not selected by the sample whit replacement of the bootstrap sample. In this manner there are multiple subsets, e.g. k from the k-fold nested CV, or calculation of X*Betas based upon independent observations, and each of these subsets can contribute to calibrate the final model. While each of these calibrations will individually have limited information, when combined following the principles of cross validation for boostrap sampling, they will collectively provide a more meaningful evaluation. This is done by the calplot() function as in

```
calplot(nested.cox.fit, wbeta=5)
```

```
##   Range of X*Beta for calibration:
## -9.020759 8.773281
##   Range of calibrated confidence intervals:
## -13.24763 10.96133
```



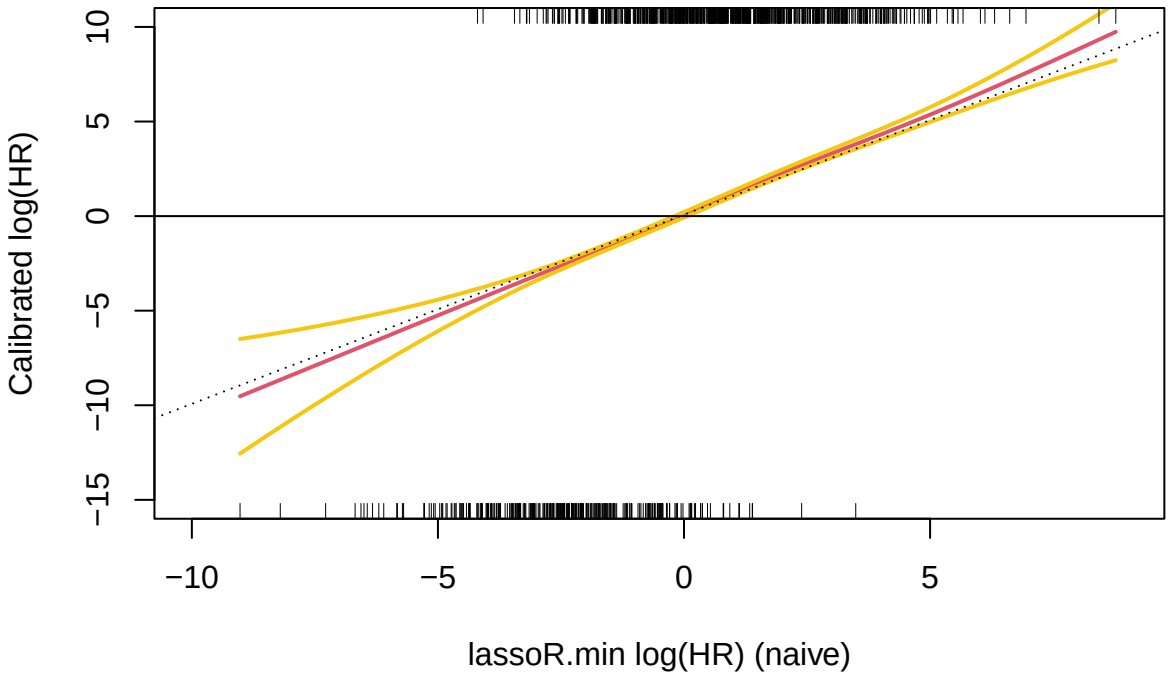
Here we see a smooth, nearly linear predicted log hazard ratio as a function of the model X* Beta from the relaxed lasso model. The bounding lines in red depict the average +/- 2 standard errors (SE) to assist in assessing meaningfulness in any deviation from the ideal identity line, and non linearities. In these curves the central region with solid lines denotes the region within the range of all the calibration spline fit, i.e. spline fits from all the different leave-out folds of the CV overlap without extrapolation. The dashed lines depict areas out of range for at least one of the leave out folds. Because spline fits can be rather uncertain when extrapolating beyond the data range, one should be more cautious in making strong

conclusions in the dashed regions of these plots. Here we see somewhat wider confidence bounds about the overall calibration curve.

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. showed that the cross validation (CV) estimates and standard errors may be biased, and should thus be viewed with caution. In particular the CV estimates may more closely estimate the expected performance measures over multiple samples than the performance of the model based upon the observed data, and usage the CV standard errors when constructing confidence intervals might be associated with mis-coverage three times the nominal non-coverage. This is discussed further in the vignette "An Overview of glmnetr".

A naive calibration curve similar to that shown above (the first figure) can also be easily gotten using the calplot function when specifying to not use the resample for construction as in

```
calplot(nested.cox.fit, wbeta=5, resample=0, xlim=c(-10,9), ylim=c(-15,10), col.term=2, col.se=7)
```

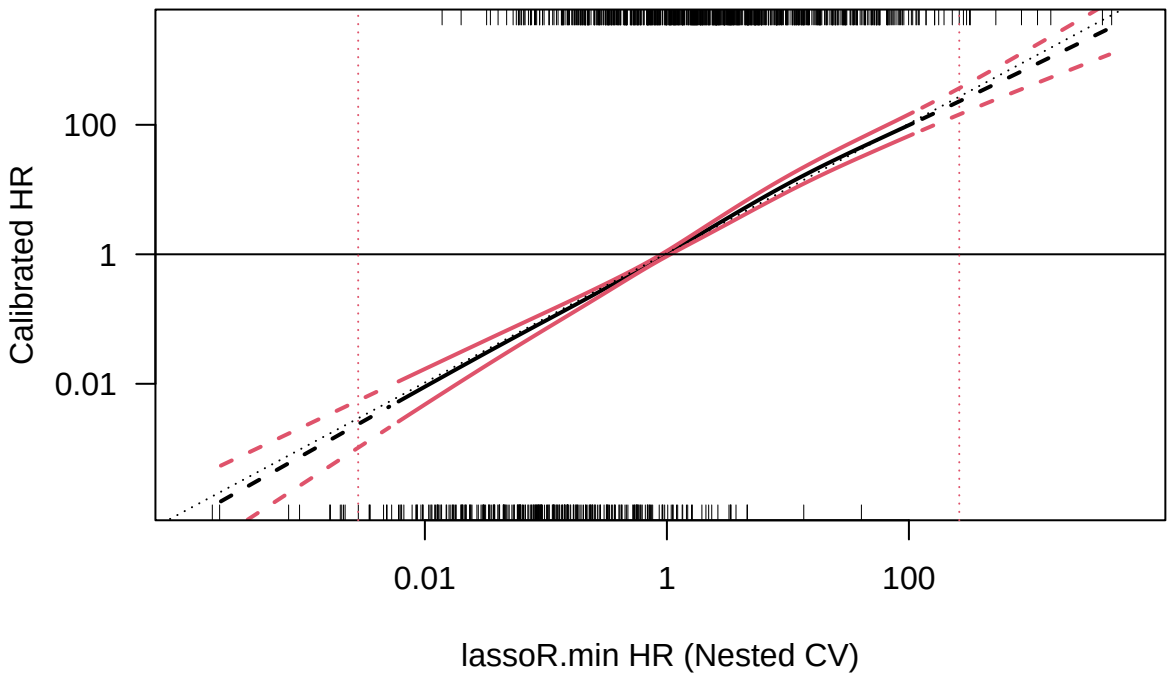


The code above using the termplot() function is provided to show our general approach for derivation of the calibration curves.

In the Nested CV figure we see two rugs, one below and one above the plotted region. The rug below depicts the model X* Beta's which are not associated with an event and the rug above depicts X* Beta's which are associated with events. When there are lots of data points it can be hard to read these rugs. One can use the vref option in calplot to draw two vertical lines where the first separates the smaller vref% of the X* Beta's form the rest, and a second which separates the larger vref% of the data. To depict the hazard ratios (HR) instead of the X*Beta for the Cox model one can use the option plothr, where one assigns a numerical value for the product between tick marks, e.g. exp(1) or 10. Combining these two options we have the example

```
calplot(nested.cox.fit, wbeta=5, vref=1, plothr=100)
```

```
##   Range of X*Beta for calibration:
## -9.020759 8.773281
##   Range of calibrated confidence intervals:
## -13.24763 10.96133
```

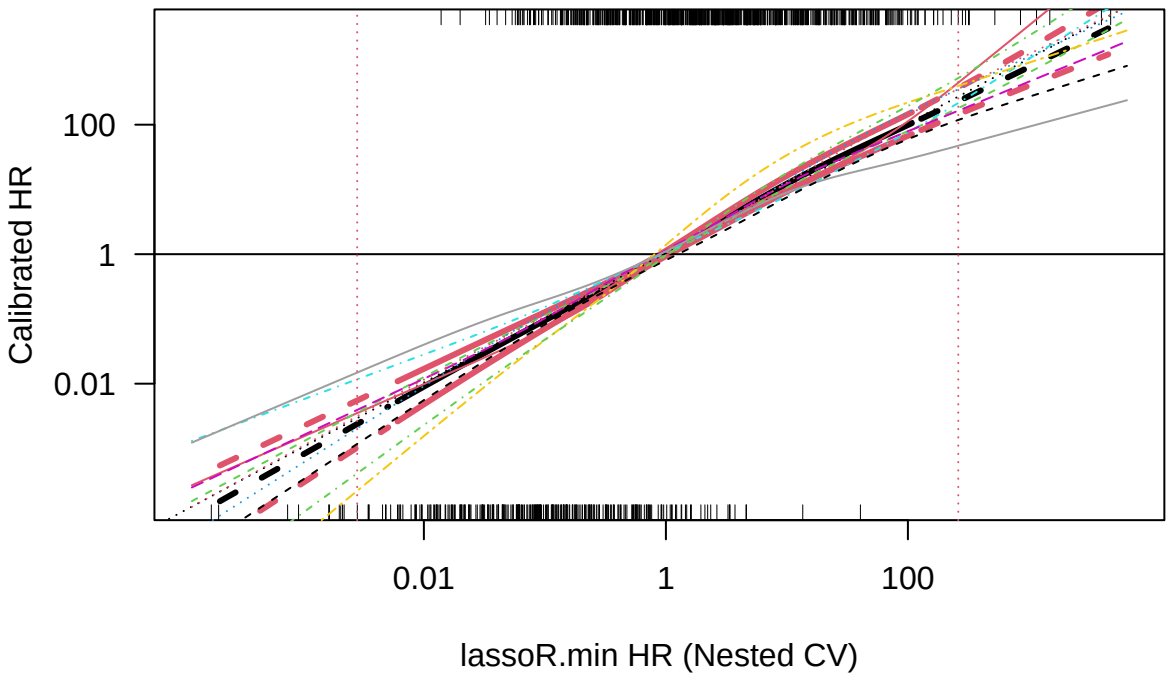


The user can also use different colors for the lines with the options col.term, col.se. One can also specify xlim and ylim in case a few data points cause an excessive amount of white space or odd aspect ratio in the plots.

To view the calibration plots form the individual leave out cross validation folds, one may specify foldplot= 1. In that this generates many figures, we omit in this vignette actually producing plots using this option specification, and instead assign plotfold=1 which overlays the individual calibration curves, albeit without the +/- 2 SE limits for the individual CV folds. The overall calibration (average of the individual CV fold calibrations) and overall +/- 2 SE limits though are maintained.

```
calplot(nested.cox.fit, 5, plotfold=1, vref=1, plothr=100)
```

```
##   Range of X*Beta for calibration:
## -9.020759 8.773281
##   Range of calibrated confidence intervals:
## -13.24763 10.96133
```



As we see from the above calls the first term in the calplot() function call is an output object form a nested.glmnetr() call. The second term, wbeta, specifies "which beta" or model is to be used for deriving the model X*Beta's. Here, as we see in the figure x-axis label, the 5 determines the relaxed lasso model. Instead of making a hard to remember key the user can leave this term unspecified and a key will be directed to the R console. The actual numbers for the different models will depend on which models are fit and so this key is dynamic.

```
calplot(nested.cox.fit)
```

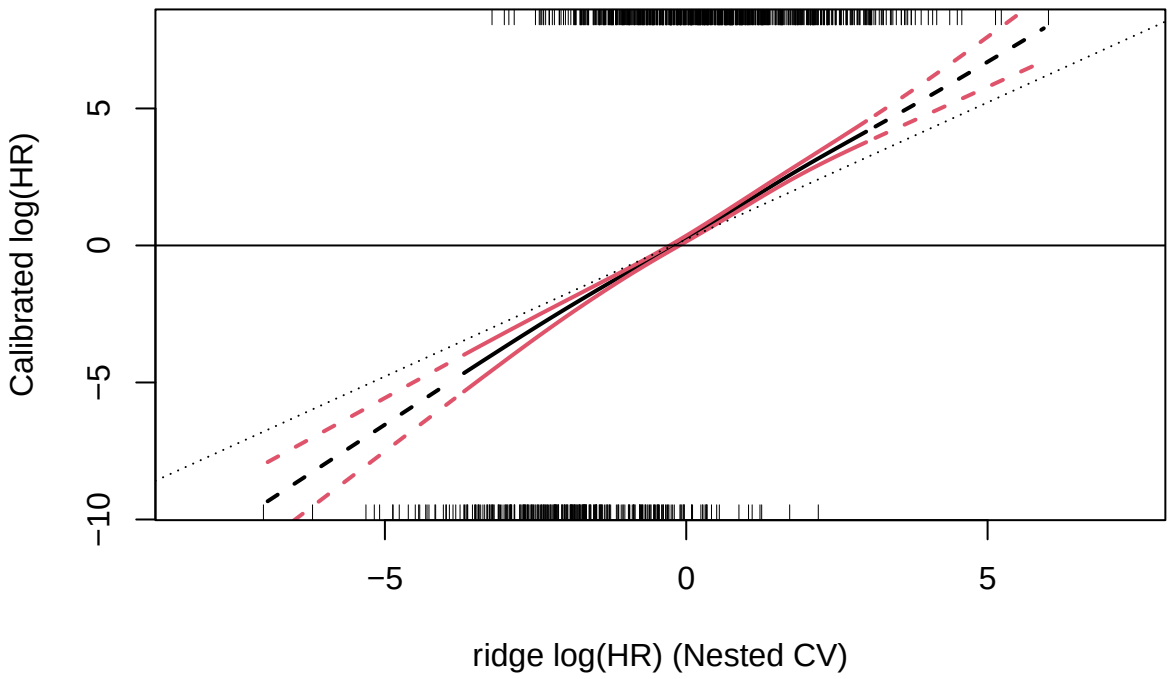
```
##  specify num for wbeta =
##                       Var num
## null            0.000000   1
## Lasso.1se       4.059254   2
## lasso.min       5.398317   3
## lassoR.1se      4.564760   4
## lassoR.min      6.174393   5
## lassoR0.1se     6.260944   6
## lassoR0.min     6.913119   7
## ridge           3.345655   8
## Lasso.1se cal   6.568479   9
## lasso.min cal   7.344196  10
## lassoR.1se cal  6.505473  11
## lassoR.min cal  7.070568  12
## lassoR0.1se cal 6.260944  13
## lassoR0.min cal 6.913119  14
## ridge cal       7.578702  15
```

```
## step.df          7.098487  16
## step.p           7.129952  17
```

From this key we read of the numers corresponding to the respective models. The variance in X*beta for the "null" model is 0 as the intercept for the Cox model is arbitrarily assinged the value of 0 for each resample model fit. From this key we see we can produce a calibration plot for the ridge regression model by setting wbeta = 8 (wbeta for "which beta"), as in

```
calplot(nested.cox.fit, 8)
```

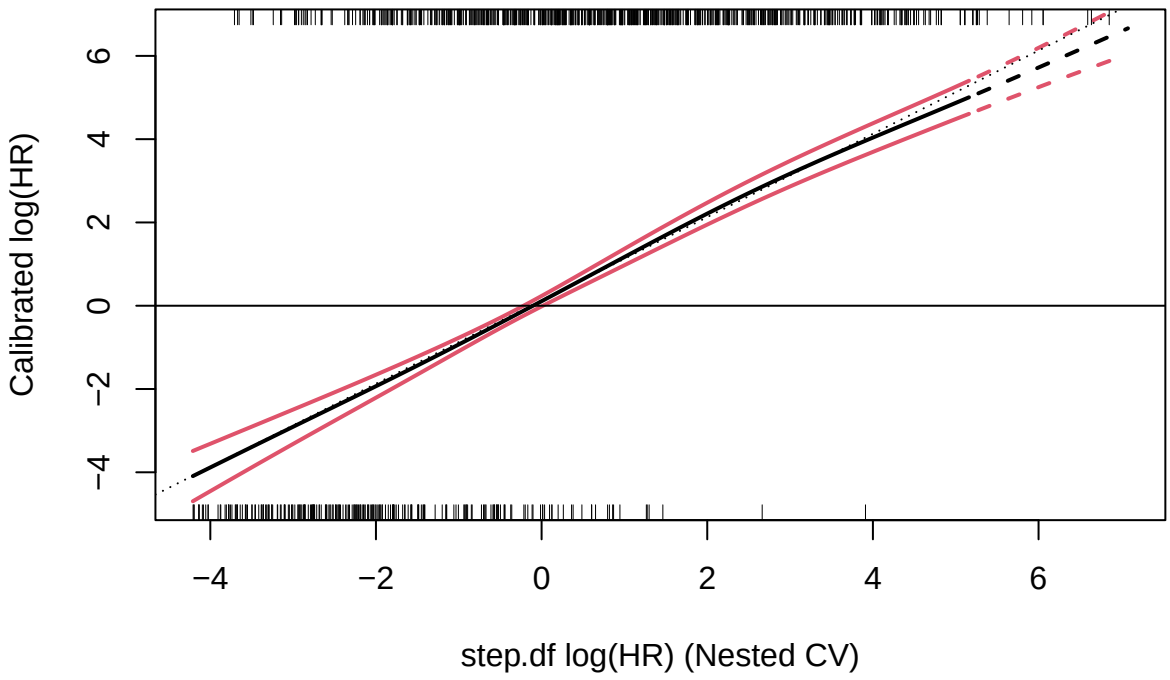
```
##   Range of X*Beta for calibration:
## -8.186478 7.32873
##   Range of calibrated confidence intervals:
## -15.51263 13.61084
```



Here we see the model is not ideally calibrated as the calibration curve largely does not include the identity line, and it requires a correction to achieve an un (less) biased estimation of the hazard ratio. Inspecting the calibration curve for a step wise regression model

```
calplot(nested.cox.fit, 16)
```

```
##   Range of X*Beta for calibration:
## -4.210835 7.079382
##   Range of calibrated confidence intervals:
## -5.646013 7.956288
```




```
##   Due to user specified xlim  67 tick marks are not displayed in rug
##   min:max xb =  -9.50800974686473 9.06934921020687
```

we see that the response is roughly linear but numerically at least there seems to be some correction for over fitting.

To obtain the numerical values used to construct these calibration plots one may specify plot=0 (or plot=2 to plot and obtain the numerical data) in list format as in

```
tmp = calplot(nested.cox.fit, 5, plot=0)
```

```
##   Range of X*Beta for calibration:
## -9.020759 8.773281
##   Range of calibrated confidence intervals:
## -13.24763 10.96133
```

```
str(tmp)
```

```
## List of 5
##  $ estimates     : num [1:101, 1:5] -9.02 -8.84 -8.66 -8.49 -8.31 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:101] "1" "2" "3" "4" ...
##   .. ..$ : chr [1:5] "plotxb" "est" "se" "lower" ...
##  $ est.resample  : num [1:10, 1:101] -8.22 -8.78 -9.93 -6.64 -8.3 ...
```

```
##  $ se.resample   : num [1:10, 1:101] 5.87 4.27 4.74 3.37 5.06 ...
##  $ lower.resample: num [1:10, 1:101] -20 -17.3 -19.4 -13.4 -18.4 ...
##  $ upper.resample: num [1:10, 1:101] 3.5212 -0.2344 -0.4553 0.0932 1.8197 ...
```

These data may be further processed by the user.

A Binomial Model

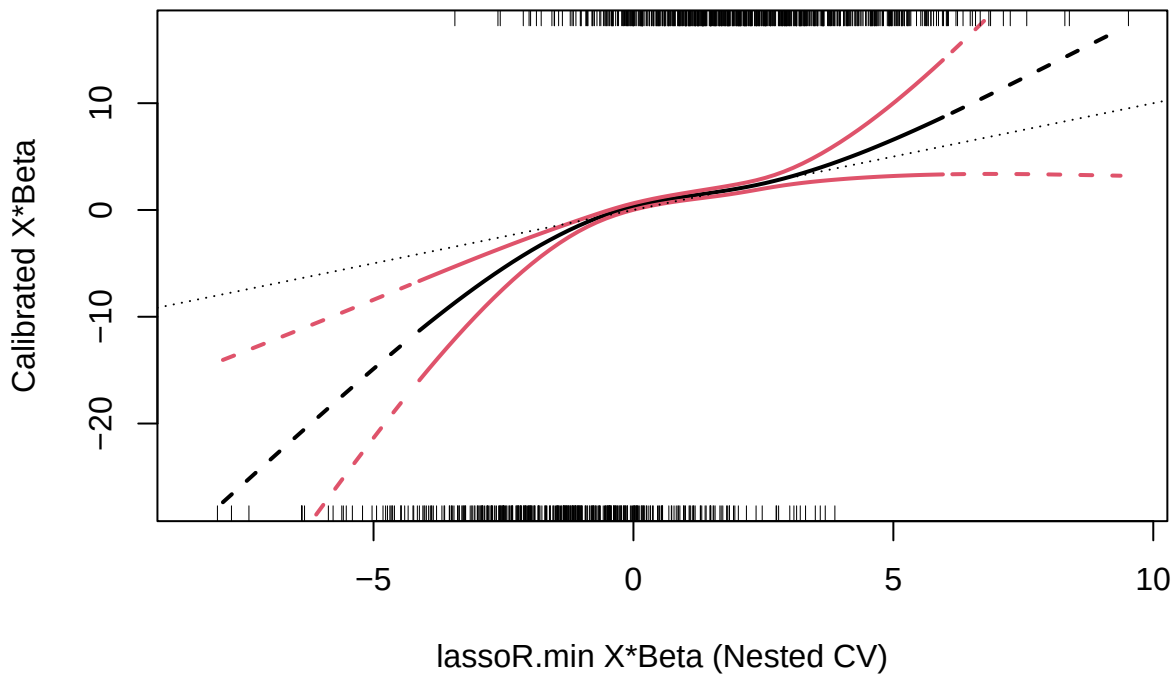
For nested.glmnetr() analyses with family = "binomial" with the call

```
yb = simdata$yb
nested.bin.fit = nested.glmnetr(xs,NULL,yb,NULL,family="binomial",
      dolasso=1, doxgb=list(nrounds=250), dorf=1, doorf=1, doann=1,
      folds_n=10, seed=219301029, track=1)
```

an example calibration plot is

```
calplot(nested.bin.fit, 5, plotfold=0)
```

```
##   Range of X*Beta for calibration:
## -8.437476 9.551432
##   Range of calibrated confidence intervals:
## -82.43294 78.38735
```

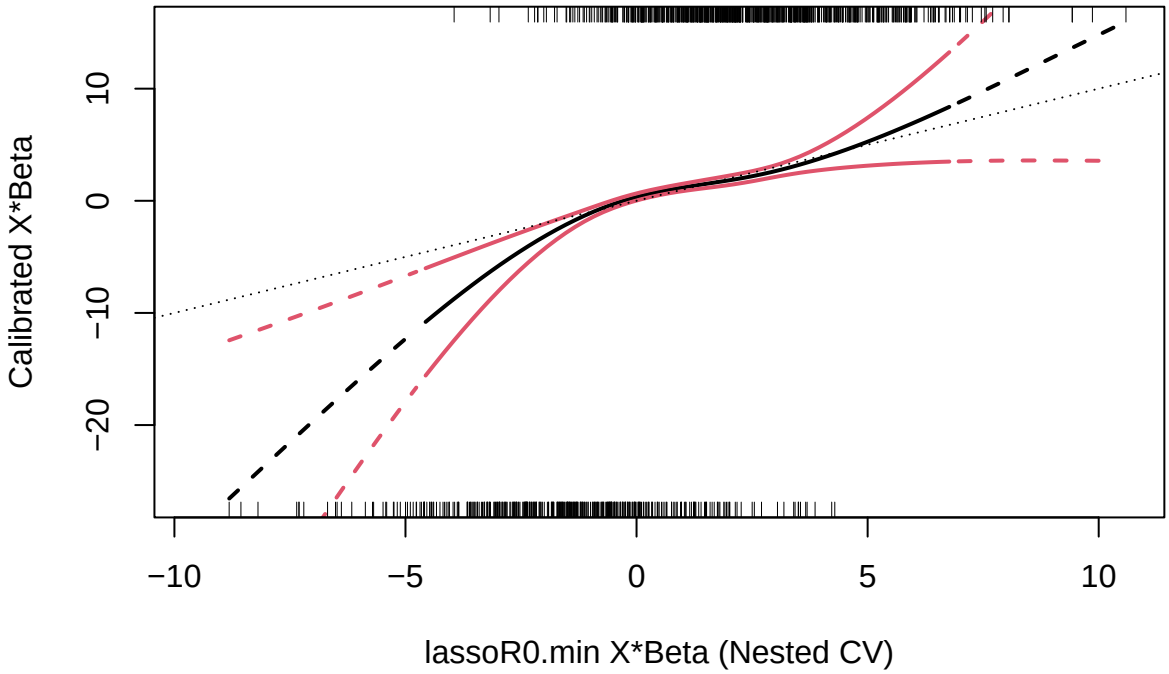


```
##   Due to user specified xlim  1 tick marks are not displayed in rug
##   min:max xb =  -8.00302508291934 9.86375009698912
```

Since these data were generated with probabilities exp(X*beta)/(1+exp(X*beta)) we want that the medal would calibrate linearly. Next we look at the "fully" relaxed model where an unpenalized model is fit based upon the non-zero terms in the fully penalized lasso model.

```
calplot(nested.bin.fit, 7, plotfold=0)
```

```
##   Range of X*Beta for calibration:
## -9.622804 10.61131
##   Range of calibrated confidence intervals:
## -89.77714 69.27437
```



This may calibrate slightly better but is not as linear as we might expect.

A Binomial Model Calibrated Using Bootstrap

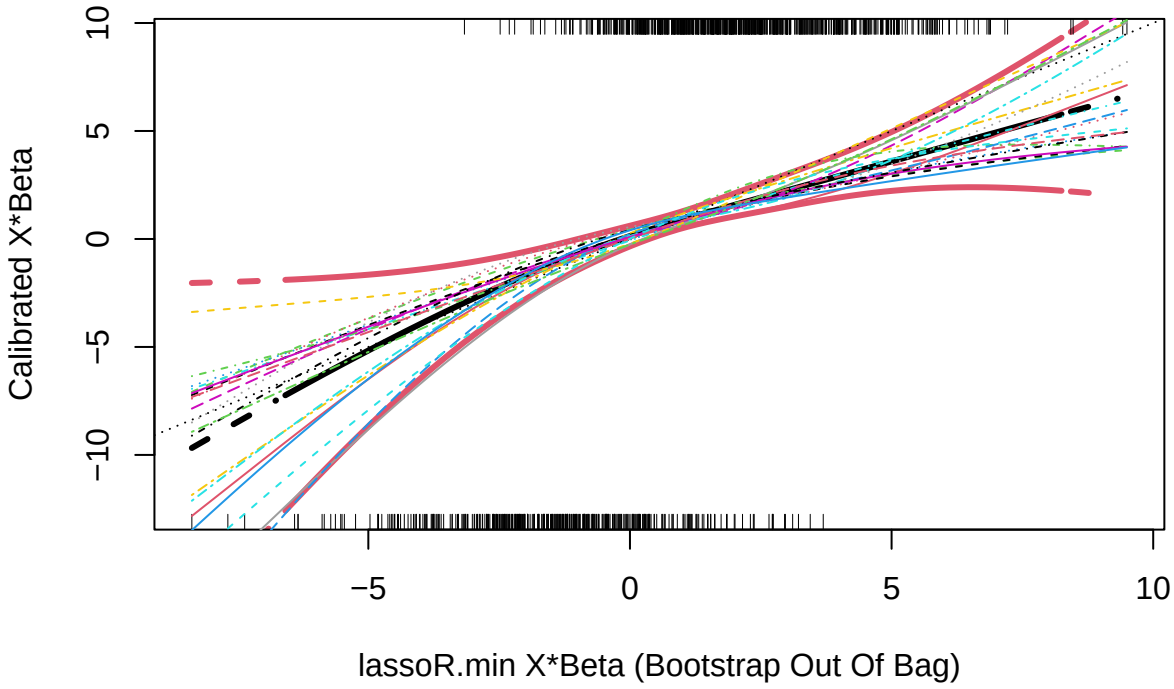
Considering this we next fit models using bootstrap, that is fit models based upon random samples from the original sample (with replacement) of same sample size as the original sample. Then we fit calibration curves for the out-of-bag sample units for each bootstrap sample fitted model, that is the elements of the original sample that are not selected by the bootstrap sample. The number of bootstrap samples for calculation is specified using the *bootstrap* option in the nested.glmnetr() call,

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

The Out-Of-Bag (OOB) calibration plots can then be constructed using the calplot() function as before,

```
calplot(nested.bin.boot.fit, 5, plotfold=1)
```

```
##   Range of X*Beta for calibration:
## -8.372612 9.498028
##   Range of calibrated confidence intervals:
## -17.8461 10.56136
```



Here we see a similar yet slightly different characteristics between the (nested) cross validation and bootstrap calibration plots. While technically the bootstrap might seem more accurate due to the confidence interval containing the ideal line (which we know applies due ot the way we simulated the data), this may as well be due to the random nature of the re-sample selection process. For the bootstrap one can run a larger number of re-samples to minimize this effect. For the cross validation one may repeat the whole process many times (see the glmnet package vignettes) and then average, but we have not done this here.

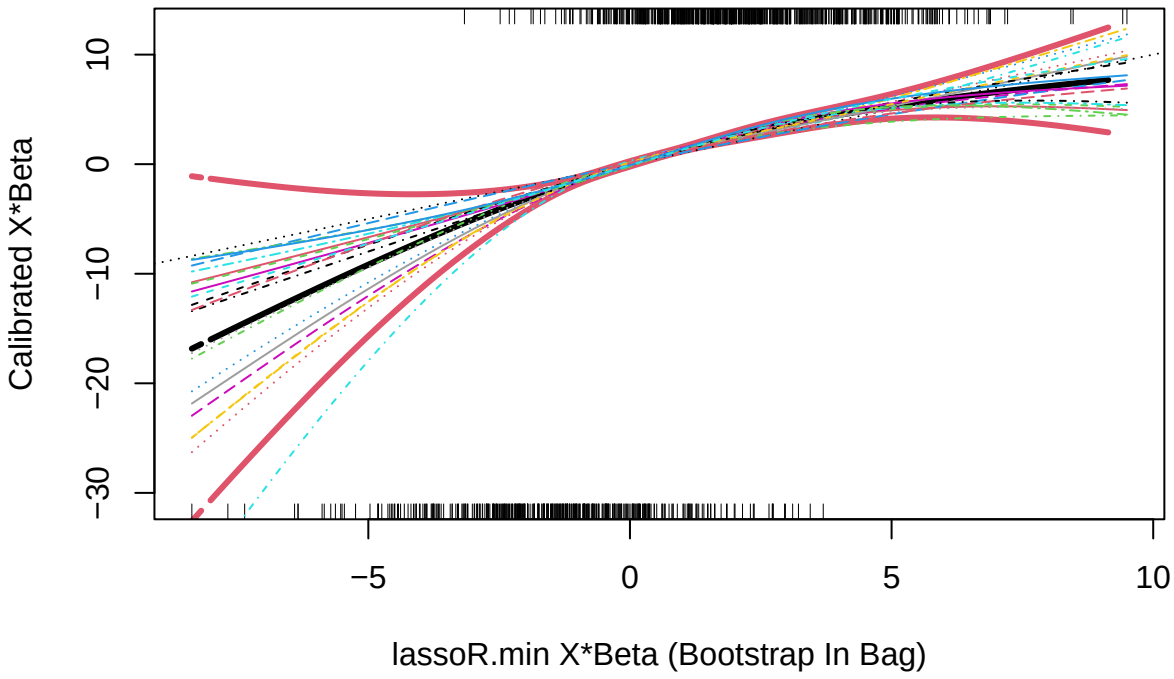
Variability in Bootstrap In-Bag Calibrations

In earlier works Austin et al. as well as Riley et al. fit models based upon bootstrap samples and then fit calibration curves based upon the in-bag data points and model XBeta (predicteds). This mimics the usual

procedure for bootstrap analysis. This can be done using the calplot() function by setting the *oob* option to 0,

```
calplot(nested.bin.boot.fit, wbeta=5, oob=0, plotfold=1)
```

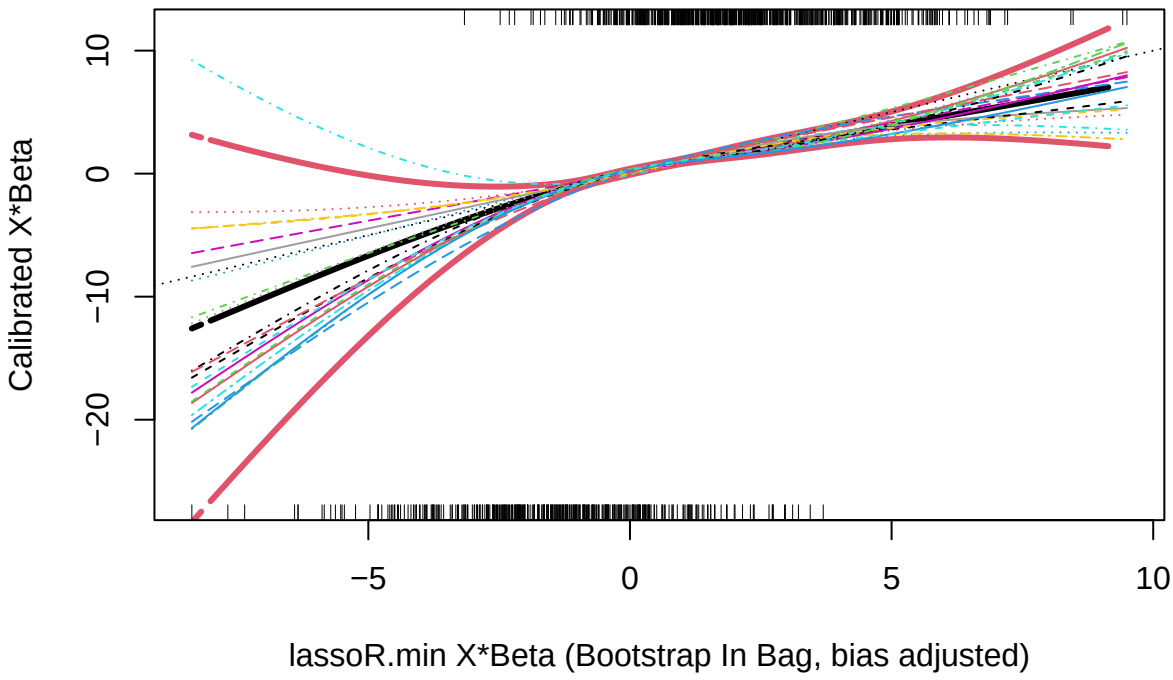
```
##   Range of X*Beta for calibration:
## -8.372612 9.498028
##   Range of calibrated confidence intervals:
## -38.63075 12.36728
```



This figure shows the variability expected in the construction of calibration curves as described by the above authors. These curves however make no adjustment for bias correction possible using the bootstrap. Making the usual bias adjustment we have (XBeta_full) - (XBeta_i - XBeta_full) or 2*XBeta_full - XBeta_i. These can be produced setting the *bootci* option to 1 as with the call

```
calplot(nested.bin.boot.fit, wbeta=5, bootci=1, plotfold=1)
```

```
##   For (bootci == 1) & (oob == 1), oob is set to 0
##   Range of X*Beta for calibration:
## -8.372612 9.498028
##   Range of calibrated confidence intervals:
```



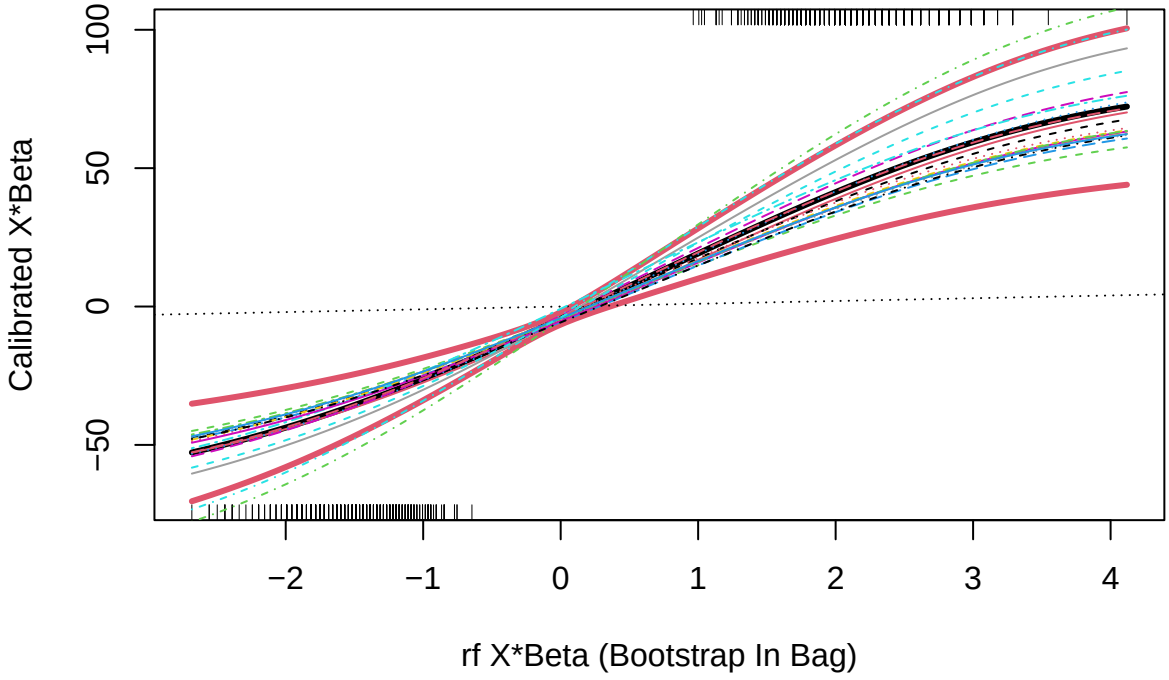
This figure differs from the simple in-bag calibrations. These two figures differ meaningfully from the out-of-bag calibration curves described above.

Bootstrap In-Bag Calibration for a Random Forest

These bootstrap plots above were for the relaxed lasso model fit. We now examine similar bootstrap calibration plots but for random forest models. The variability in the in-bag bootstrap calibratin curves for the random forest model is depicted in the graph

```
calplot(nested.bin.boot.fit, wbeta=16, bootci=0, oob=0, plotfold=1)
```

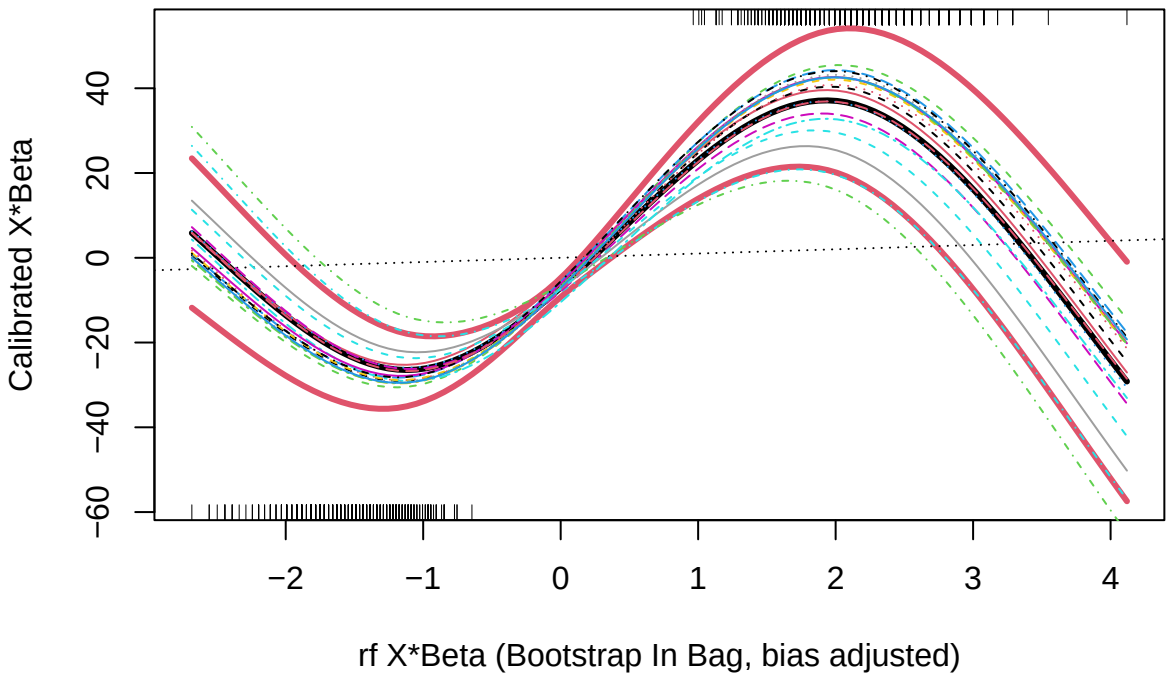
```
##   Range of X*Beta for calibration:
## -2.682732 4.119037
##   Range of calibrated confidence intervals:
## -77.7793 108.1106
```



Here we see a strong deviation from the identity line. The bias corrected bootstrap calibration curves

```
calplot(nested.bin.boot.fit, wbeta=16, bootci=1, plotfold=1)
```

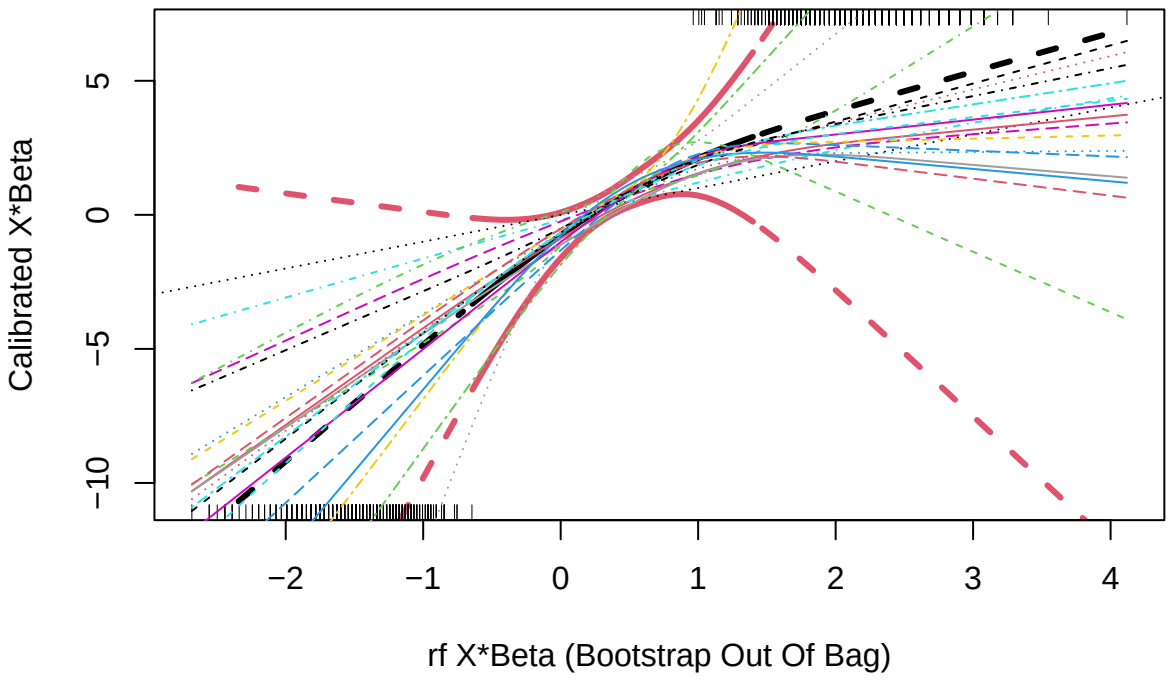
```
##   For (bootci == 1) & (oob == 1), oob is set to 0
##   Range of X*Beta for calibration:
## -2.682732 4.119037
##   Range of calibrated confidence intervals:
```



have a more unusual and unexpected deviation from the ideal calibration curve of the identity line. Returning to the out-of-bag calibration bootstrap plots

```
calplot(nested.bin.boot.fit, wbeta=16, plotfold=1)
```

```
##   Range of X*Beta for calibration:
## -2.682732 4.119037
##   Range of calibrated confidence intervals:
## -34.91789 42.46664
```



we see that these too are highly variable but more consistent with the ideal calibration curve, the identity line, than the in-bag calibration curves.

While these two in-bag calibration curve sets strongly depart from the ideal of the identity line, the out-of-bag curves while possibly highly variable were consistent as a group with the ideal of the identity line. The bootstrap out-of-bag calibration curves have a clearer basis and depict more reasonable findings. In a sense this is not unexpected. When fitting machine learning models one typically evaluates model fit based upon the out-of-bag data points and not the in-bag data points. A difficulty here could be that the calibration curves, calculated at particular values for XBeta, do not describe a well defined parameter in terms of the disturbution function. Potentially as well the fitting process for machine learning models might not lend it self to the assumptions typically used to support bootstrap inferences. In particular as we see here when fitting random forest models, with the multiple repeat observations in the bootstrap resamples the model refits are overly optimistic, more strongly overfit and give more extreme XBeta as evidenced in these figures. We present the in-bag and biased adjusted bootstrap calibration curves not to promote their usage but to 1) show the variability in model fit and 2) how they may give poor inferences for a simple dataset. The user may want to generate other datasets of different known form and see how the bootstrap performs for their data.

A Normal (Gaussian Errors) Model

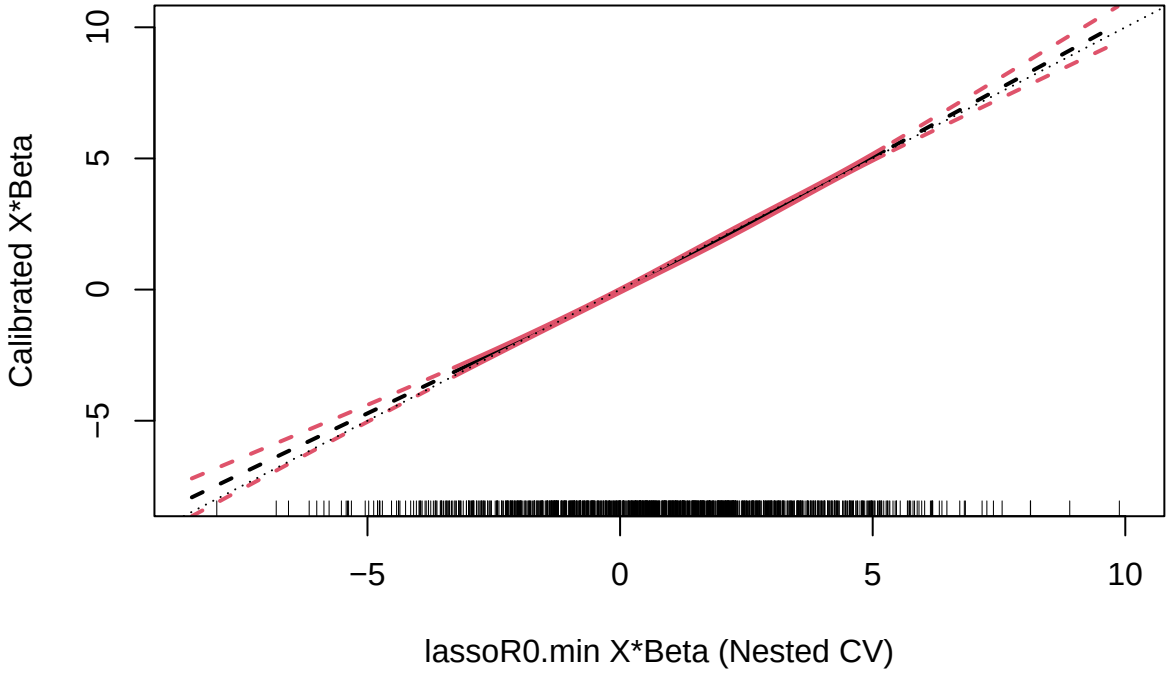
First calculating the numerical summaries of prediction performance

```
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)
```

and then plotting

```
calplot(nested.gau.fit, wbeta=7)
```

```
##   Range of X*Beta for calibration:
## -8.47665 10.03264
##   Range of calibrated confidence intervals:
## -9.677822 11.94403
```




```
##   Due to user specified xlim  2 tick marks are not displayed in rug
##   min:max xb =  -8.68249320568155 10.2389399048293
```

we see a small but probably not significant deviation from the ideal calibration line, the identity function.

Perspective

The summary and calibration plot functions used here do not address all needed for model validation and calibration but do allow a meaningful and un (or minimally) biased summary of model fits. The original outcome variable and X*betas are stored as vectors or matrices in the output with names y_, xbetas (full model) and xbetas.cv (cross-validation) and xbetas.boot.oob and betas.boot.inb (bootstrap out-of-bag and in-bag) allowing the user to further inspect model fits and to perform other means of calibration. For cross-validation analyses the fold information is contained in the output object in the vector foldid. For bootstrap analyses the first column of xbetas.boot.oob and betas.boot.inb describe the replication of the bootstrap sampling process and the second columns the sequential index for the data point in the input dataset.

References

Austin PC, Steyerberg EW. Graphical assessment of internal and external calibration of logistic regression models by using loess smoothers. Stat Med. 2014; 33(3):517-35. doi: 10.1002/sim.5941 .

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 .

Riley RD, Pate A, Dhiman P, Archer L, Martin GP, Collins GS. Clinical prediction models and the multiverse of madness. BMC Med. 2023; 21(1):502. doi: 10.1186/s12916-023-03212-y .