Calibration of Machine Learning Models in glmnetr

Walter K. Kremers, Mayo Clinic, Rochester MN

20 April 2024

The Package

The "An Overview of glmnetr" vignette hows 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. conordance, 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, similar to how we assessed other model performance metrics.

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, amonsg other information, summaries for the linear calibration slopes and intercepts as in

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

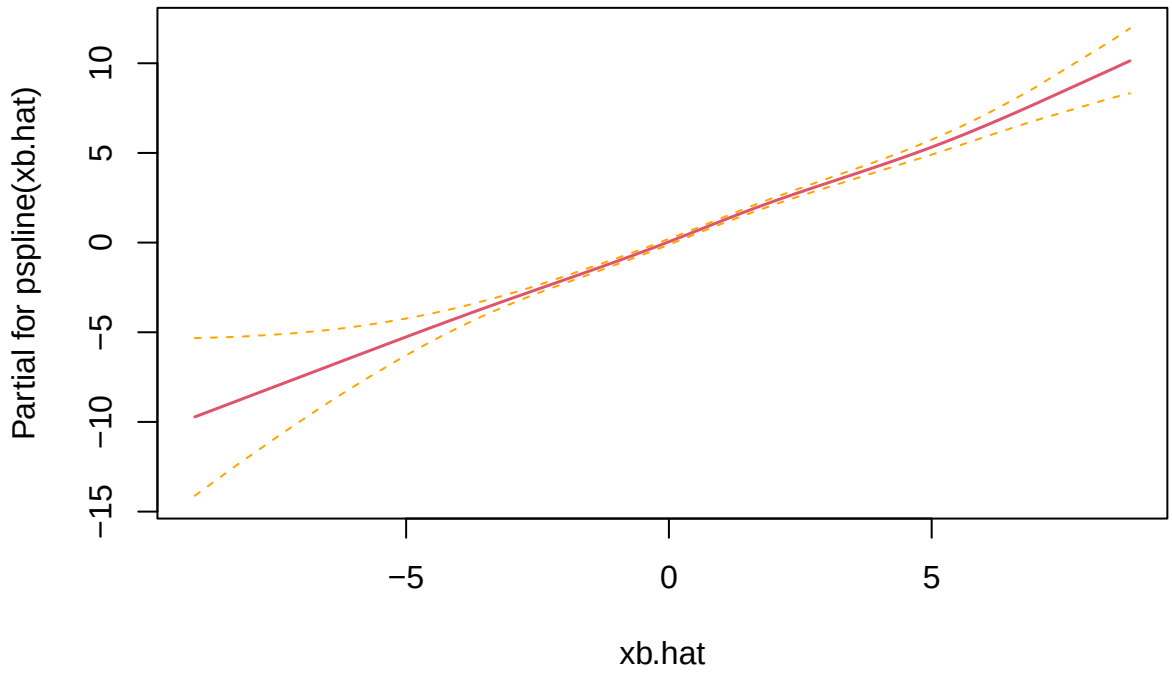
An initial calibration consideration was made in the overview vignette by regressing observed outcomes on the predicteds from the final model based upon the relaxed lasso. This regression was made using splines, in particular the pspline() function from within a coxph() call, as in

```
# 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
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
```

followed by plotting with

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



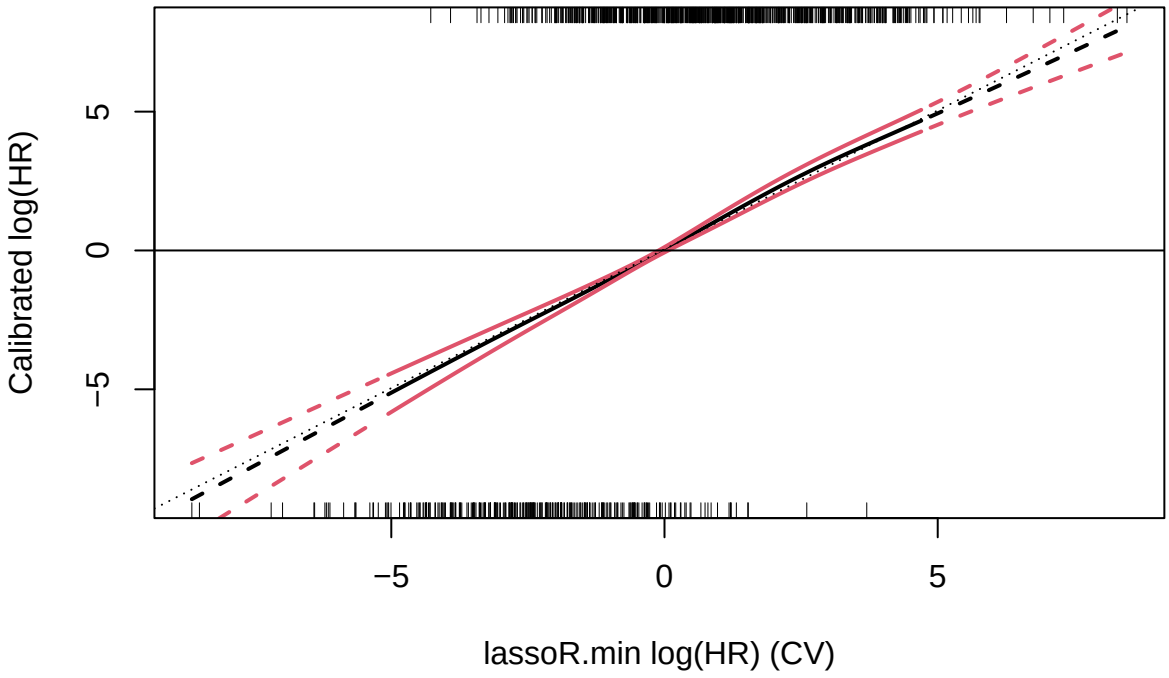
The spline fits may help to understand potential nonlinearities in the model. Here we see, a clibration line which is not far from linear. Still, as noted in the "An Overview of glmnetr" vignette, because the same

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 cross validation

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. fold, of the outer loop. In this manner there are k subsets (presuming k-fold CV) of X*Betas calculated based upon "new" data, 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, they will collectively provide a more meaningful evaluation. This is done by the calplot() function as in

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

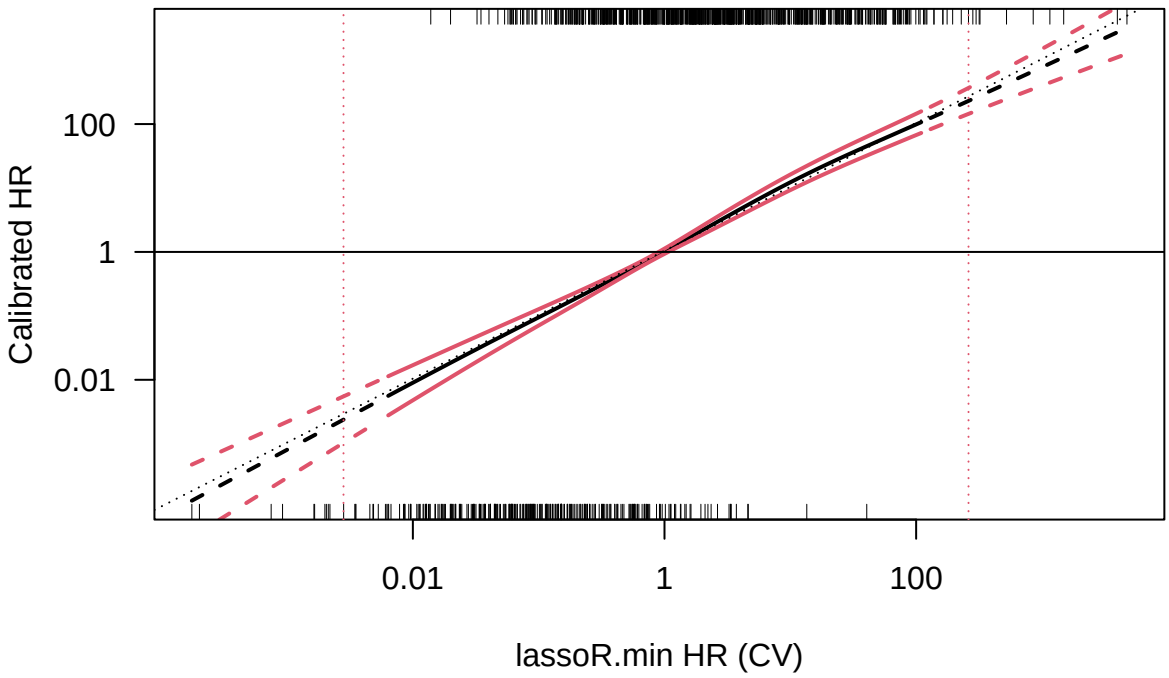


Here we see a smooth, nearly linear predicted log hazard ratio as a function of the model X* Beta from the relaxed lasso model. The bounding lines in red depict the average +/- 2 standard errors (SE) to assist in assessing meaningfulness in any deviation from the ideal identity line, and non linearities. 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.

In this 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

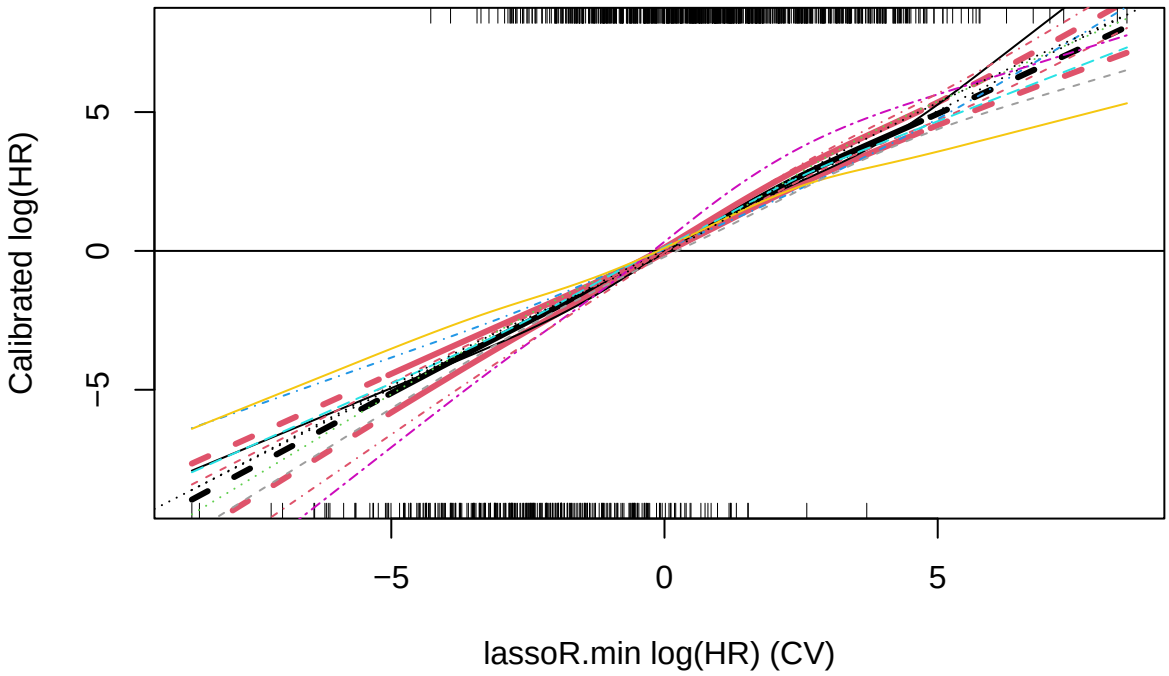
```
calplot(nested.cox.fit, wbeta=5, vref=1, plothr=100)
```



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=2 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=2)
```



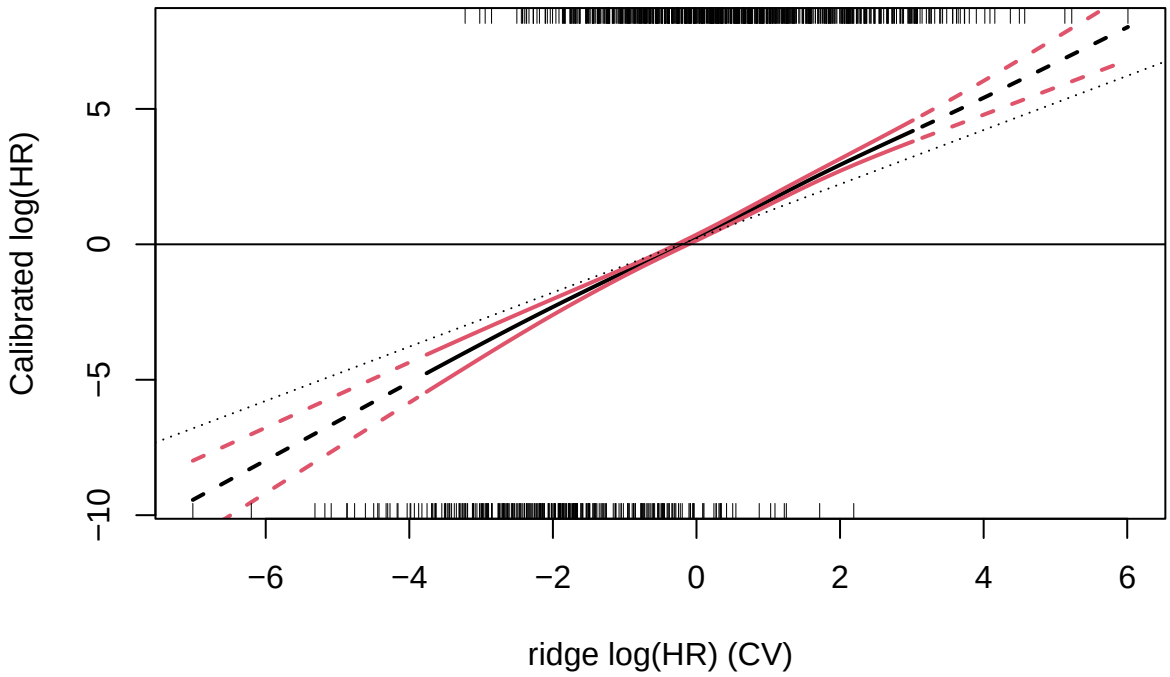
The first term in the calplot() function call is an output object form a nested.glmnetr() call. The second term, wbeta, specifies which "beta" or model is to be used for deriving the model X*Beta's. Here, as we see in the figure x label, the 5 determines the relaxed lasso model. Instead of making a hard 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     4.059266   2
## lasso.min     5.412081   3
## lassoR.1se    4.564760   4
## lassoR.min    6.174396   5
## lassoR0.1se   6.260944   6
## lassoR0.min   6.913119   7
## ridge         3.345655   8
## step.df       7.086523   9
## step.p        7.118197  10
## aic           0.000000  11
```

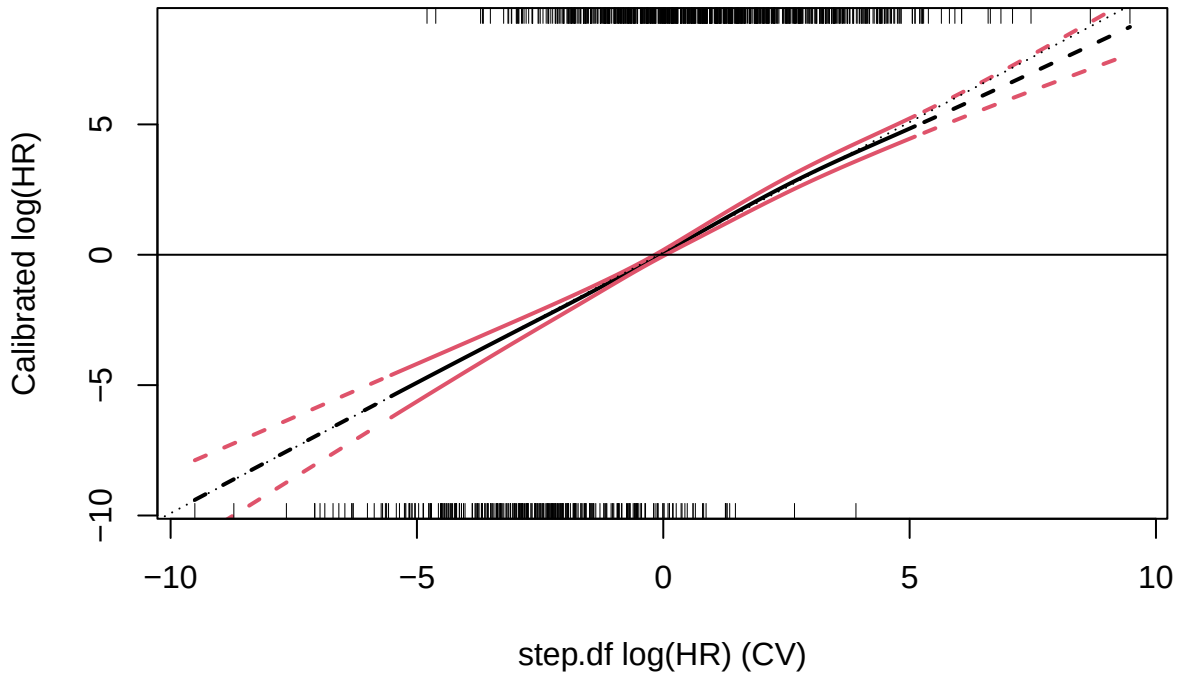
From this key we see we can produce a calibration plot for the ridge regression model, for example, with the code

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



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, 9)
```



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)
str(tmp)
```

```
## List of 5
##  $ estimates: num [1:101, 1:5] -8.65 -8.48 -8.31 -8.14 -7.97 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:101] "1" "2" "3" "4" ...
##   .. ..$ : chr [1:5] "plotxb" "est" "se" "lower" ...
##  $ est.cv   : num [1:10, 1:101] -7.91 -8.41 -9.49 -6.38 -7.97 ...
##  $ se.cv    : num [1:10, 1:101] 5.4 3.98 4.4 3.13 4.69 ...
##  $ lower.cv : num [1:10, 1:101] -18.7 -16.4 -18.3 -12.6 -17.4 ...
##  $ upper.cv : num [1:10, 1:101] 2.879 -0.455 -0.694 -0.13 1.417 ...
```

These data may be further processed by the user.

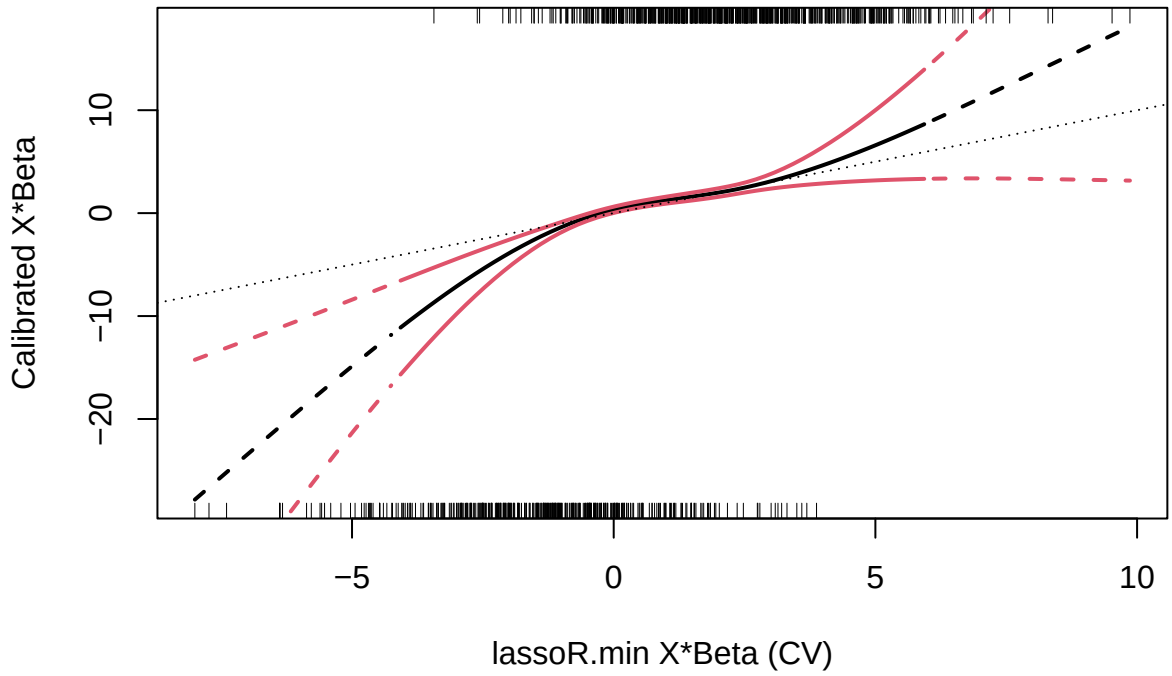
A Binomial Model

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, doann=1, ensemble=c(1,0,0,0, 0,1,0,1),
      folds_n=10, seed=219301029, track=1)
```

an example calibration plot is

```
calplot(nested.bin.fit, 5)
```



Note, the simulated data for this and the Cox model fit above involved deviations from the standard linear model assumptions so a deviation from a perfect calibration with slope of 1 is not unexpected.

Perspective

The calibration plots shown here do not address all needed for model validation and calibration but do provide a meaningful, un (or minimally) biased summary of model fits, thus are supportive of model evaluation.