Calibration of Machine Learning Models in glmnetr

Walter K. Kremers, Mayo Clinic, Rochester MN

10 July 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. 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 boostrap sampling, 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

```
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                 0.2446    1.0742          0.8705         45.4
## LASSO minR                0.2434    0.9790          0.8701         17.6
## LASSO minR.GO             0.2410    0.9320          0.8691         16.2
## Ridge                     0.2244    1.2850          0.8626         99.0
##                       Naive DevRat Naive Concordance Non Zero
## LASSO min                   0.1696            0.8794       42
## LASSO minR                  0.1720            0.8793       22
## LASSO minR.GO               0.1728            0.8796       20
## Ridge                       0.1718            0.8822       99
##
##                       Ave DevRat Ave Slope Ave Concordance Ave Non Zero
## Stepwise df tuned         0.2501    0.9590          0.8738         14.7
## Stepwise p tuned          0.2527    0.9677          0.8747         14.8
##                       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

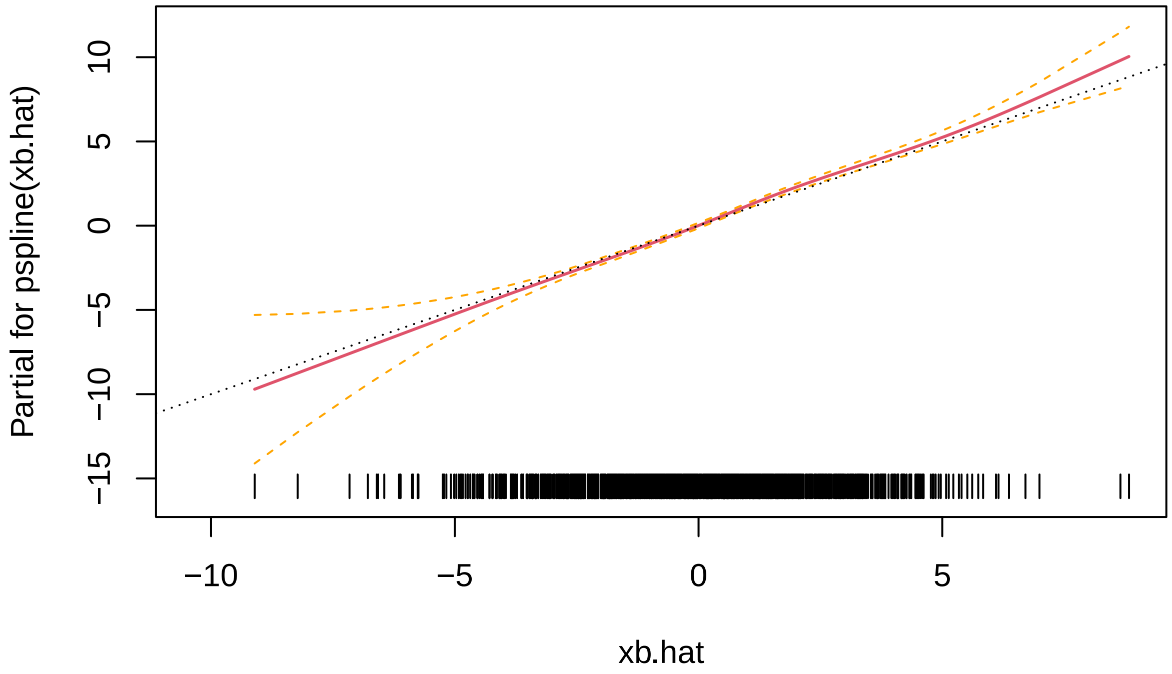
```r
# Get predicteds from CV relaxed lasso model embedded in nested CV outputs & Plot
xb.hat = predict( object=nested.cox.fit , xs_new=xs, lam=NULL, gam=NULL, comment=FALSE)
# Fit a spline to xb.hat uisng coxph, and plot
#library( survival )                  ## load survival package for Cox model fits
fit1 = coxph(Surv(yt, event) ~ pspline(xb.hat))
```

```r
summary(fit1)
```

```
## Call:
## coxph(formula = Surv(yt, event) ~ pspline(xb.hat))
##
##   n= 1000, number of events= 698
##
##                          coef  se(coef) se2     Chisq   DF   p
## pspline(xb.hat), linear 1.068 0.03323  0.03323 1033.02 1.00 1.2e-226
## pspline(xb.hat), nonlin                           4.62 3.04  2.1e-01
##
##              exp(coef) exp(-coef) lower .95 upper .95
## ps(xb.hat)3  7.032e+00  1.422e-01 6.919e-01 7.148e+01
## ps(xb.hat)4  4.946e+01  2.022e-02 7.686e-01 3.182e+03
## ps(xb.hat)5  3.476e+02  2.877e-03 1.408e+00 8.582e+04
## ps(xb.hat)6  2.401e+03  4.165e-04 4.617e+00 1.249e+06
## ps(xb.hat)7  1.477e+04  6.772e-05 2.429e+01 8.978e+06
## ps(xb.hat)8  9.365e+04  1.068e-05 1.611e+02 5.446e+07
## ps(xb.hat)9  8.562e+05  1.168e-06 1.465e+03 5.003e+08
## ps(xb.hat)10 4.690e+06  2.132e-07 7.977e+03 2.758e+09
## ps(xb.hat)11 2.593e+07  3.857e-08 4.369e+04 1.538e+10
## ps(xb.hat)12 2.464e+08  4.059e-09 3.956e+05 1.535e+11
## ps(xb.hat)13 2.639e+09  3.789e-10 3.593e+06 1.938e+12
## ps(xb.hat)14 2.889e+10  3.461e-11 2.522e+07 3.311e+13
##
## Iterations: 4 outer, 16 Newton-Raphson
##      Theta= 0.7408233
## Degrees of freedom for terms= 4
## Concordance= 0.879  (se = 0.005 )
## Likelihood ratio test= 1504  on 4.04 df,   p=<2e-16
```

followed by plotting with

```
termplot(fit1,term=1,se=TRUE, rug=TRUE)
abline(a=0,b=1,lty=3)
```

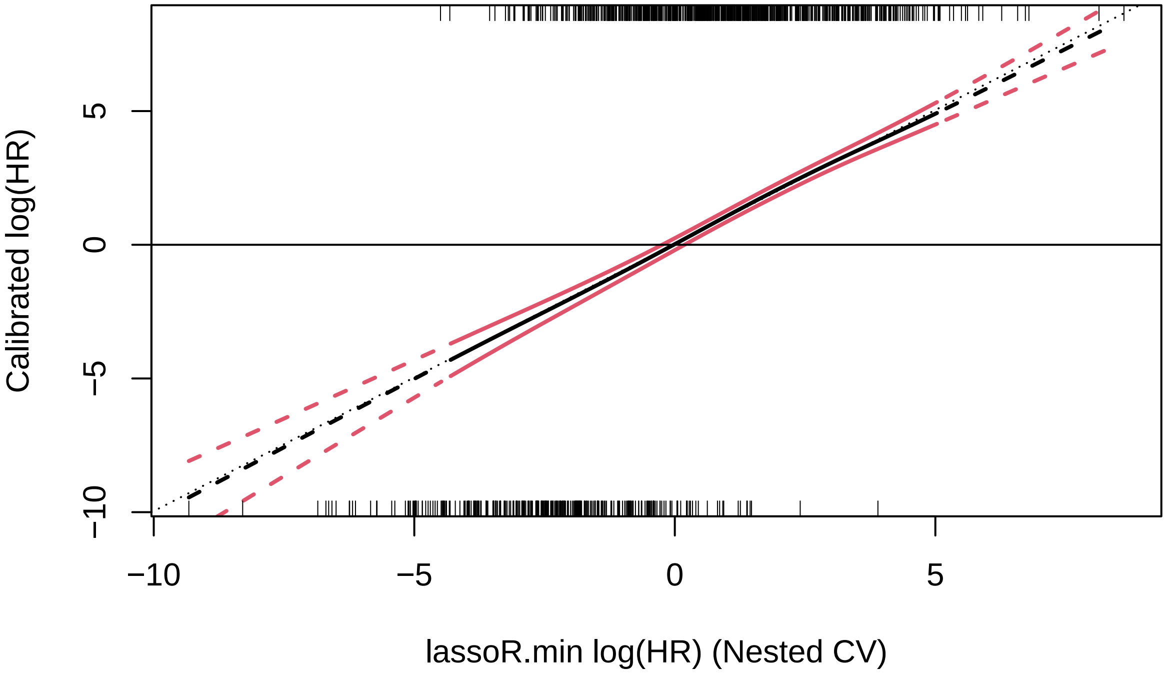


The spline fits may help to understand potential nonlinearities in the model. Here we see, a clibration line which is not far from linear. Still, as noted in the “An Overview of glmnetr” vignette, because the same data are used for model evaluation as well as model derivation, it is hard to put much 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 teh sample wiht reaplcement of teh bootstrap sample. In this manner there are multiple subsets, e.g. k from the k-fold nested CV, or calcualtion 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

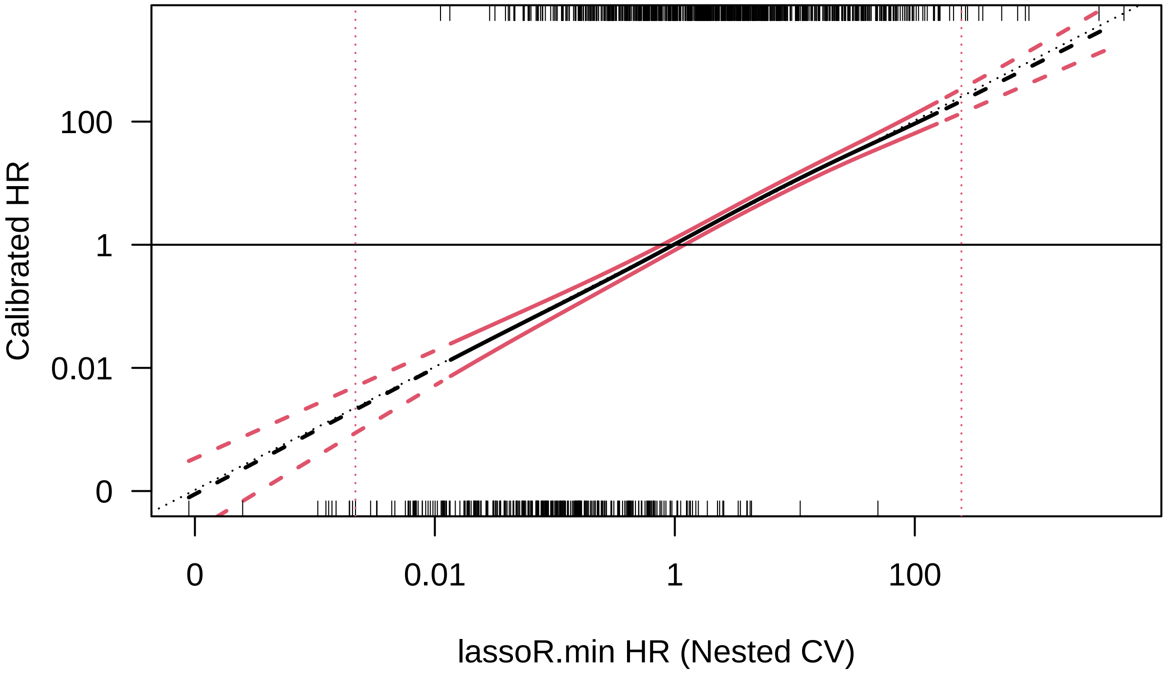
```
calplot(nested.cox.fit, wbeta=5)
```



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

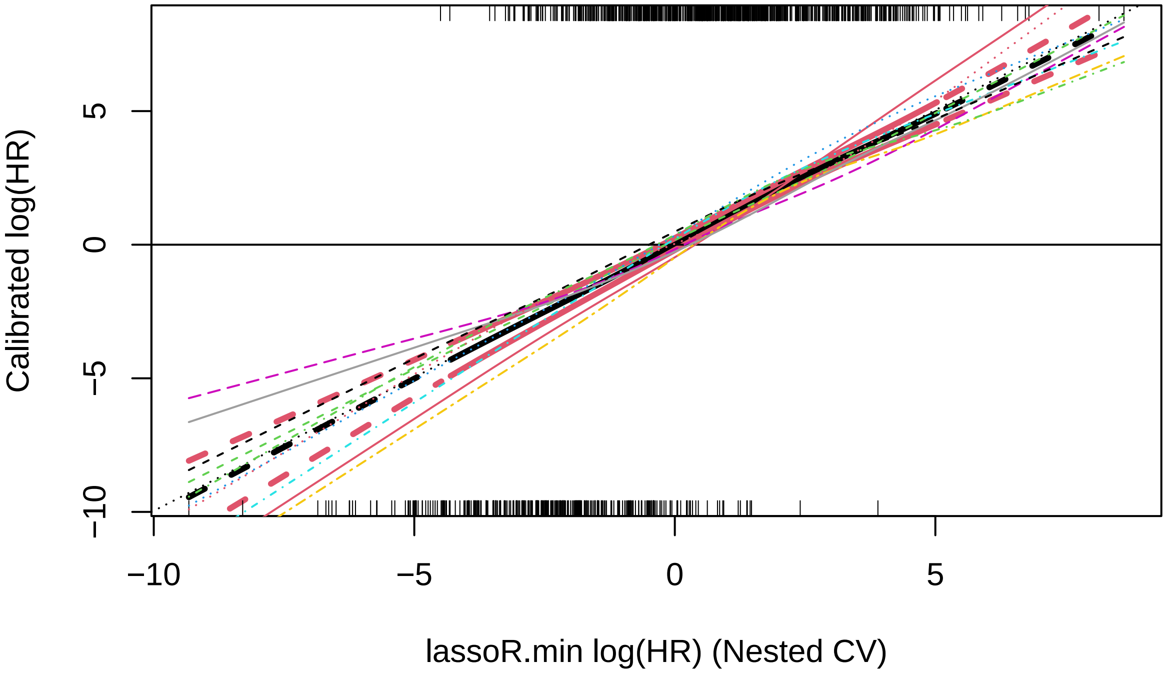
```
calplot(nested.cox.fit, wbeta=5, vref=1, plothr=100)
```



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=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)
```



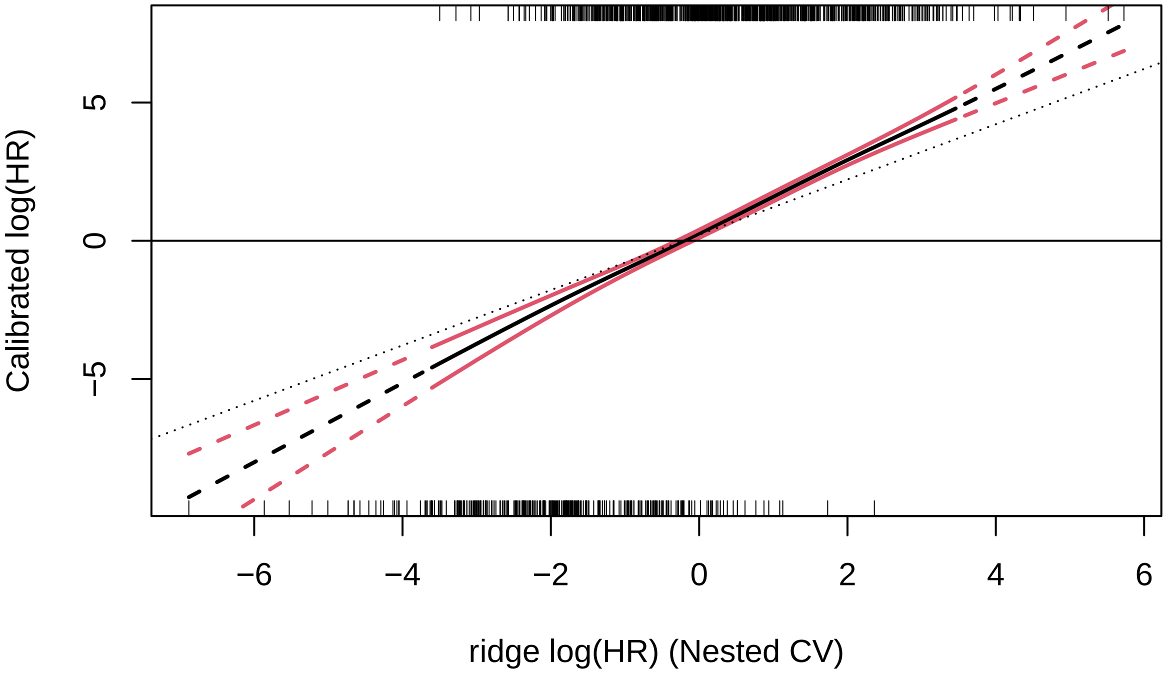
As we see from the above calls the first term in the calplot() function call is an output object form a nested.glmnetr() call. The second term, wbeta, specifies "which beta" or model is to be used for deriving the model X*Beta's. Here, as we see in the figure x-axis label, the 5 determines the relaxed 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      3.924673   2
## lasso.min      5.284658   3
## lassoR.1se     4.752238   4
## lassoR.min     6.308573   5
## lassoR0.1se    5.399642   6
## lassoR0.min    6.939080   7
## ridge          3.316922   8
## Lasso.1se cal  6.476655   9
## lasso.min cal  7.333156  10
## lassoR.1se cal 6.365883  11
## lassoR.min cal 6.960332  12
## lassoR0.1se cal 5.399642  13
## lassoR0.min cal 6.939080  14
## ridge cal       7.550848  15
## step.df         7.694102  16
## step.p          7.692468  17
```

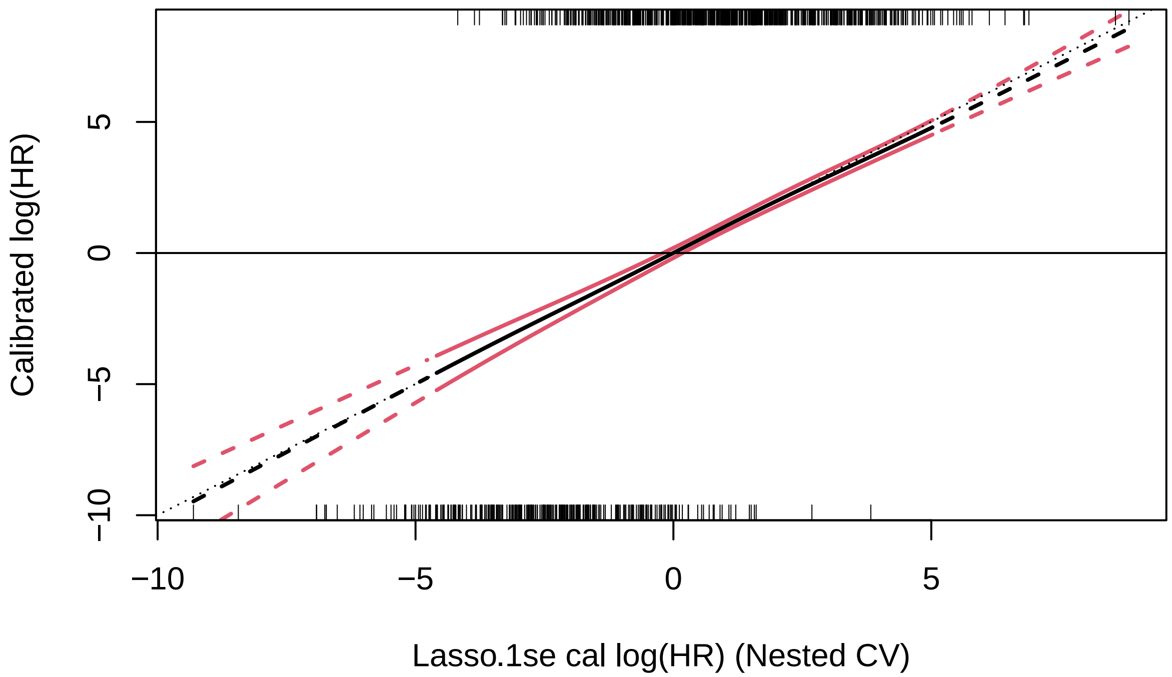
From this key we read of the numers corresponding to the respective models. The variance in X*beta for the "null" model is 0 as the intercept for the Cox model is arbitrarily assinged the value of 0 for each resample model fit. From this key we see we can produce a calibration plot for the ridge regression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

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



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] -9.33 -9.15 -8.97 -8.79 -8.61 ...
##   ..- 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] -12.01 -8.88 -9.77 -11.33 -5.74 ...
##  $ se.resample   : num [1:10, 1:101] 5.08 4.26 5.28 4.36 3.69 ...
##  $ lower.resample: num [1:10, 1:101] -22.2 -17.4 -20.3 -20.1 -13.1 ...
##  $ upper.resample: num [1:10, 1:101] -1.855 -0.366 0.79 -2.606 1.631 ...
```

These data may be further processed by the user.

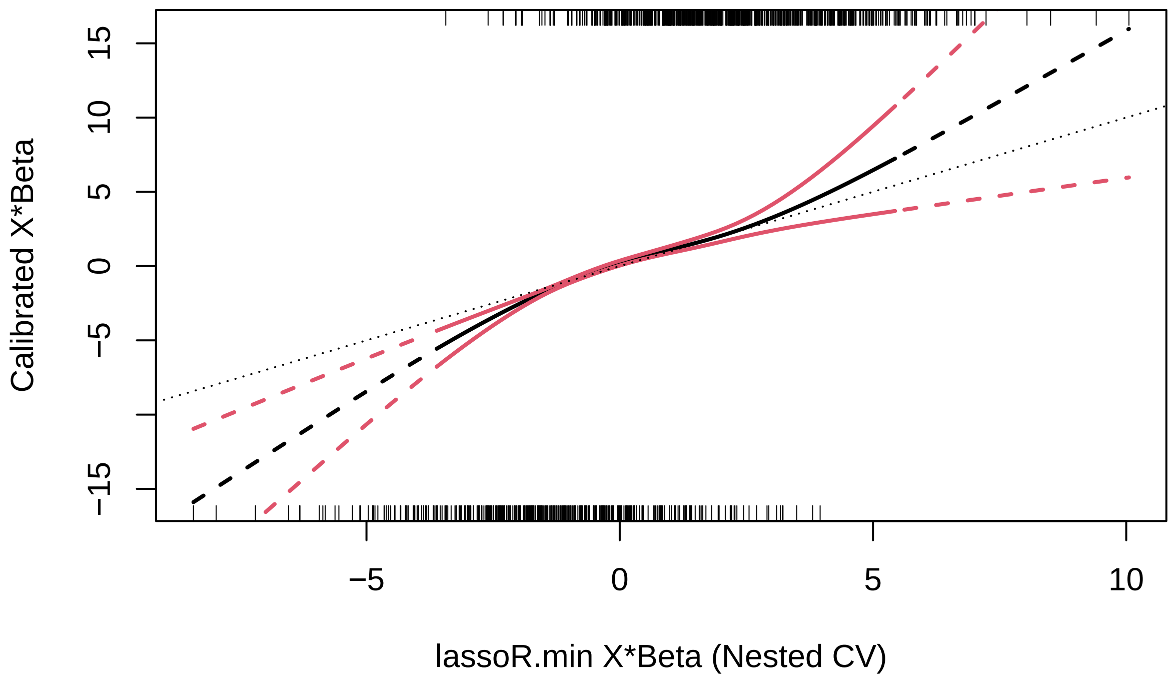
A Binomial Model

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, ensemble=1,
      folds_n=10, seed=219301029, track=1)
```

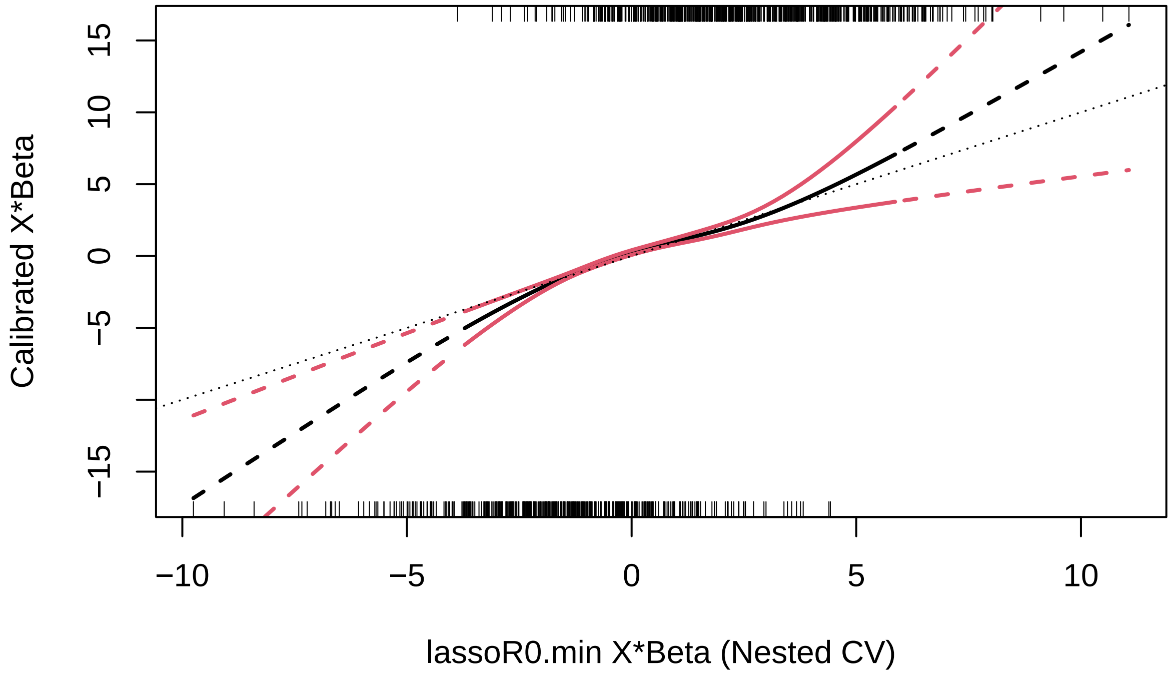
an example calibration plot is

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



Since these data were generated with probabilities exp(X*beta)/(1+exp(X*beta)) we would expect 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)
```

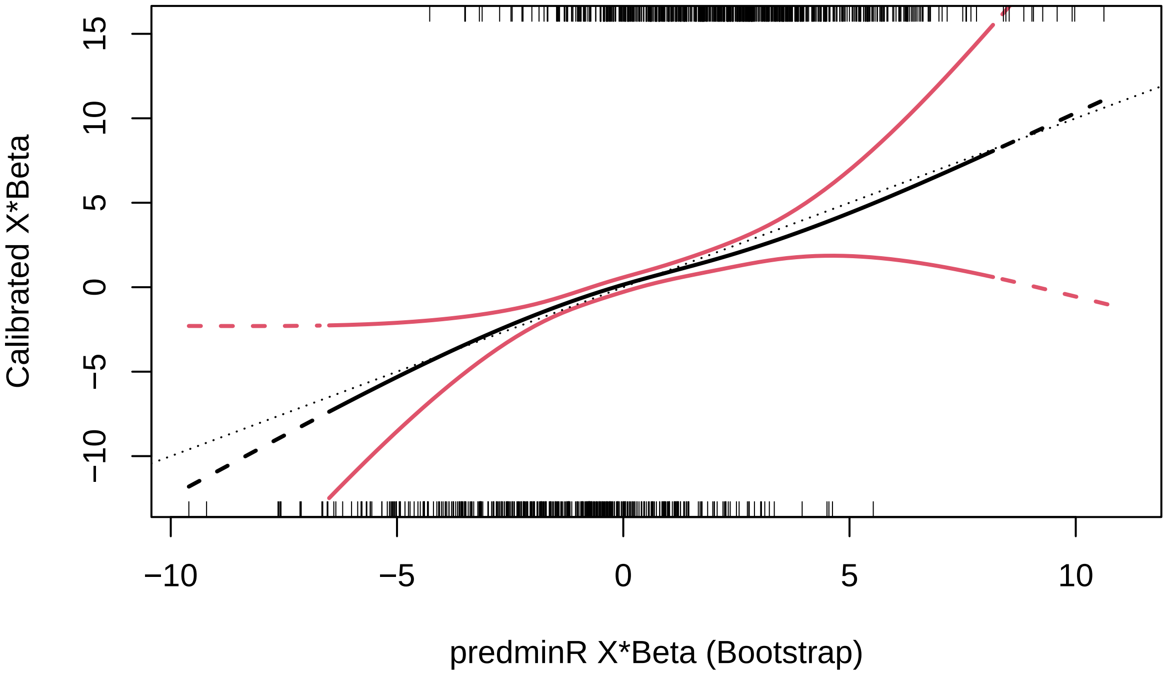


This may calibrate better but is not as linear as we might expect. Considering this we next fit models using bootstrap, that is fit models based upon random samples from the original sample (with replacement) of size the same as the original sample. Then we fit calibration curves for the out-of-bag sample units for each bootstrap sample, that is the elements of the original sample that are not selected by the bootstrap sample. This is done by specifying the number of bootstrap samples for calculation with the bootstrap option in the calplot() call. First we perform a nested cross validation,

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

with resulting plot

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



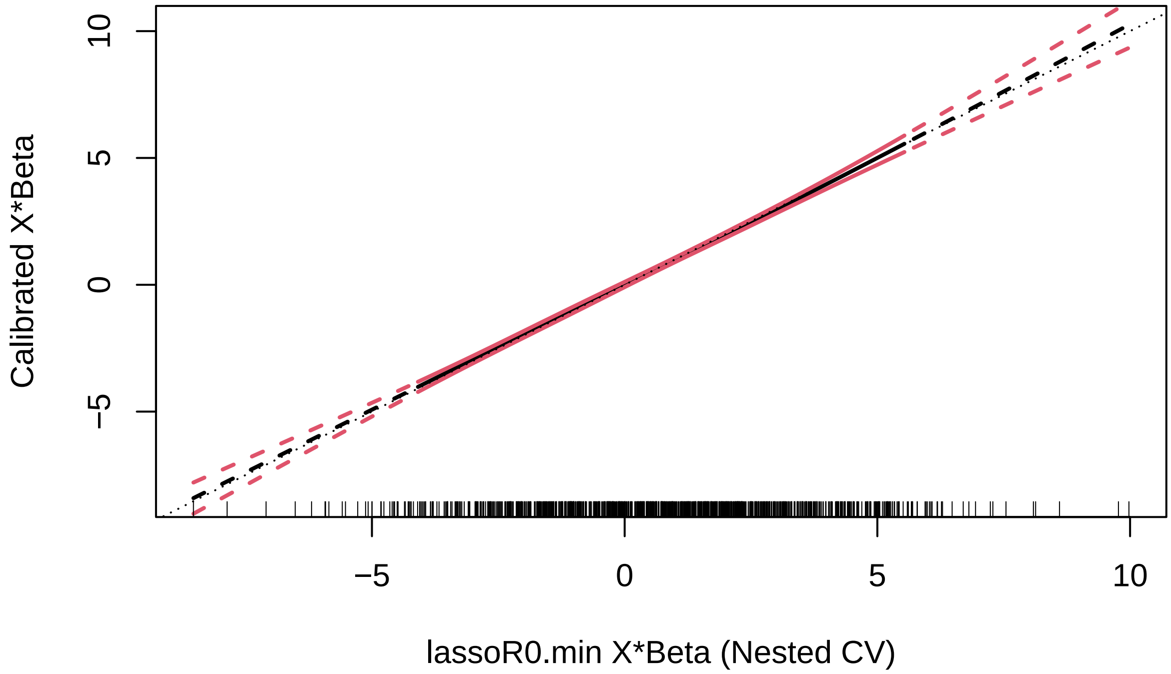
Knowing by the data construction that the true relation between predictors and outcome is linear, it appears that the bootstrap might be better at approximating this linearity.

A Normal (Gaussian errors) Model

First calculating the numerical summaries of prediction performance

and then plotting we see

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



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.