

# An Overview of glmnet

Walter K. Kremers, Mayo Clinic, Rochester MN

22 March 2024

## The Package

The `nested.glmnet()` function of the ‘glmnet’ package allows the user to fit multiple machine learning models on a common dataset with a single function call allowing an efficient comparison of different modeling approaches. Additionally this function uses cross validation (CV) to estimate model performances for these different modeling approaches. As most of these machine learning models choose hyperparameters informed by a cross validation or some sort of out of bag (OOB) performance measure, the `nested.glmnet()` function provides model performance estimates based upon either a nested cross validation (NCV) or analogous approach. Measures of model performance include concordances for survival time and binomial outcomes and R-squares for quantitative numeric outcomes, as well as deviance ratios, i.e.  $1 - \text{deviance}(\text{model}) / \text{deviance}(\text{nullmodel})$ , 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. Whereas linear calibration does not exhaust the needs of calibration considerations, it does provide a first high level insight. As the purpose of the `nested.glmnet()` function is to not only describe performance but to derive the models, each of the fitted models as well as performance measures are stored in a single output object.

The `nested.glmnet()` function fits cross validation informed Relaxed lasso, ridge, gradient boosting machine (‘xgboost’), Random Forest (‘RandomForestSRC’), 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 `do_ncv=0`.

As with the ‘glmnet’ package, tabular and graphical summaries can be generated using the `summary` and `plot` functions. Use of the ‘glmnet’ 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 datasets, for example when the design matrix is not of full rank, ‘glmnet’ may have very long run times when fitting the relaxed lasso model, from our experience when fitting Cox models on data with many predictors and many patients, making it difficult to get solutions from either `glmnet()` or `cv.glmnet()`. This may be remedied with the ‘`path=TRUE`’ option when calling `cv.glmnet()`. In the ‘glmnet’ package we always take an approach like that of `path=TRUE`.

When fitting not a relaxed lasso model but an elastic-net model, then the R-packages ‘nestedcv’ <https://cran.r-project.org/package=nestedcv>, ‘glmnetSE’ <https://cran.r-project.org/package=glmnetSE> or others may provide greater functionality when performing a nested CV.

## Data requirements

The basic data elements for input to the *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 the data for the same sampling unit.

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

## An example dataset

To demonstrate usage of *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.

```
# Extract simulated survival data  
xs = simdata$xs      # matrix of predictors  
y_ = simdata$yt     # vector of survival times  
event = simdata$event # indicator of event vs. censoring
```

Inspecting the predictor matrix we see

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

```
## 1000 100
```

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

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

```
# Inspect the first few rows and some select columns  
print(xs[1:10,c(1:12,18:20)])
```

```
##      X1 X2 X3 X4 X5 X6 X7 X8 X9 X10 X11 X12      X18      X19      X20
## [1,]  1  1  0  0  0  0  0  0  0  1  0  1  0.1513225 -0.4034383  0.35250844
## [2,]  1  0  0  0  1  0  0  1  0  0  0  0 -1.1610480  0.5533030  0.14578868
## [3,]  1  0  0  1  0  0  1  0  0  0  0  0 -0.3292269  0.3086399 -0.48443836
## [4,]  1  1  0  0  0  0  0  0  0  1  0  0  2.0635214 -0.5500741 -0.02173104
## [5,]  1  0  0  0  1  0  0  1  0  0  0  0  0.3905722 -0.6836452 -0.37643201
## [6,]  1  0  1  0  0  0  0  0  1  0  0  0 -0.2397597  1.6909447  0.49599945
## [7,]  1  0  1  0  0  0  0  1  0  0  0  0 -0.5592424  0.2314638 -0.53198341
## [8,]  1  0  0  1  0  0  0  0  0  0  1  0 -1.0050514  0.5319574  0.54287646
## [9,]  1  0  0  1  0  0  0  0  0  0  1  0  1.2548034  0.8213164  0.17067691
## [10,] 1  0  0  0  1  0  0  0  1  0  0  0 -0.3079151 -0.6105910 -0.88711869
```

## Performance of cross validation (CV) informed relaxed lasso model

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 train 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 SRDM ( Simon R, Radmacher MD, Dobbin K, McShane LM. Pitfalls in the Use of DNA Microarray Data for Diagnostic and Prognostic Classification. J Natl Cancer Inst (2003) 95 (1): 14-18. <https://academic.oup.com/jnci/article/95/1/14/2520188> ). 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” dataset 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.glmnet()` 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(465783345)
nested.cox.fit = nested.glmnet(xs, NULL, y_, event, family="cox",
                              dolasso=1, dostep=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 which elicits warning messages. We suppress these warnings here. The first term in the call to `nested.glmnet()`, `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 `y_`, is the outcome variable for the linear regression or logistic regression model and the time of event or censoring in case of the Cox model, and finally the fourth term is the event indicator variable for the Cox model taking the value 1 in case of an event or 0 in case of censoring at time `y_`. The fourth term would be `NULL` for either linear or logistic regression. If one sets `track=1` the program will update progress in the R console, else for `track=0` it will not. We recommend setting `track=1` when running the program interactively. Depending on the size of the dataset 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, degrees 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. Hyperparameters considered for stepwise regression are degrees of freedom (df) and p, the p-value for entry into the regression equation, as discussed by JWHT (James, Witten, Hastie and Tibshirani, An Introduction to Statistical Learning with applications in R, 2nd

ed., Springer, New York, 2021). Performance measures include deviance 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 dataset.

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

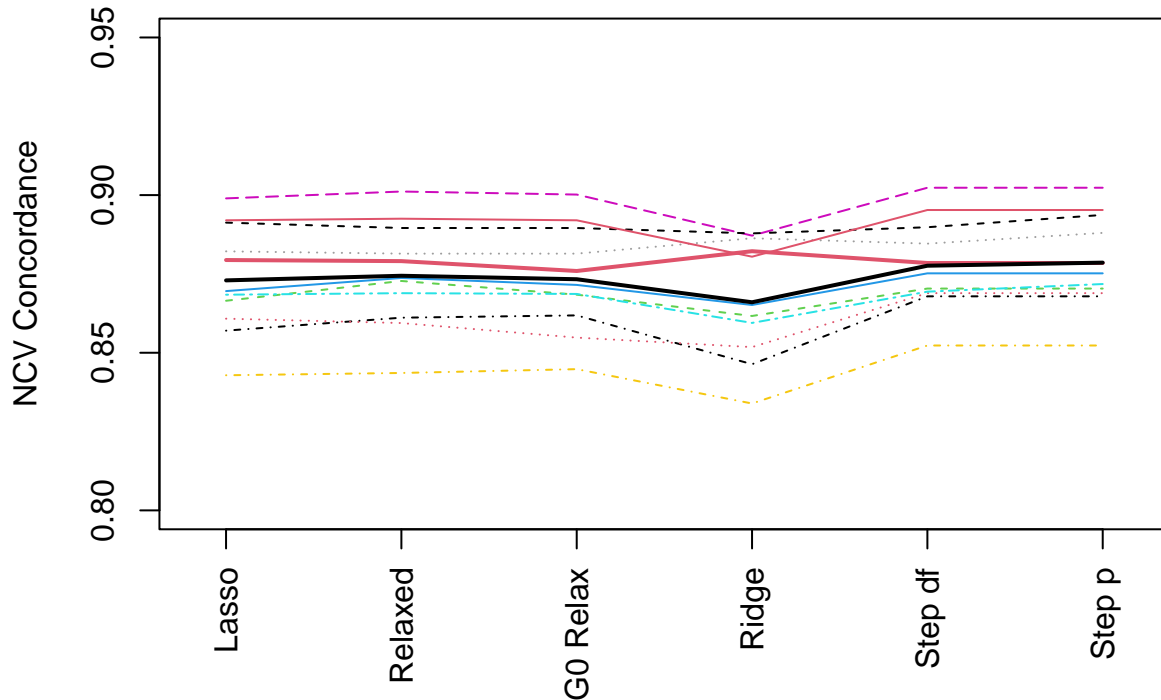
## Sample information including number of records, events, number of columns in
## design (predictor, X) matrix, and df (rank) of design matrix:
##           family              n          nevent
##           cox              1000          698
##           xs.columns      xs.df      null.dev/nevent
##           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 min           0.2452  1.0702           0.8730           48.0
## LASSO minR          0.2470  1.0083           0.8744           20.9
## LASSO minR.GO       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           19
## LASSO minR.GO       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, not unexpectedly, that the concordances estimated from the nested CV are smaller than the concordances naively calculated using original dataset. 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 dataset 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 predicted, i.e. the XBeta or the log(hazard ratio). (Many machine learning fitting routines use as predicted the hazard ratio for “cox” model generalizations or the probability for “binomial” data. We give predicted 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 model performance informed by cross validation
plot(nested.cox.fit, type="agree", ylim=c(0.8,0.95))
```



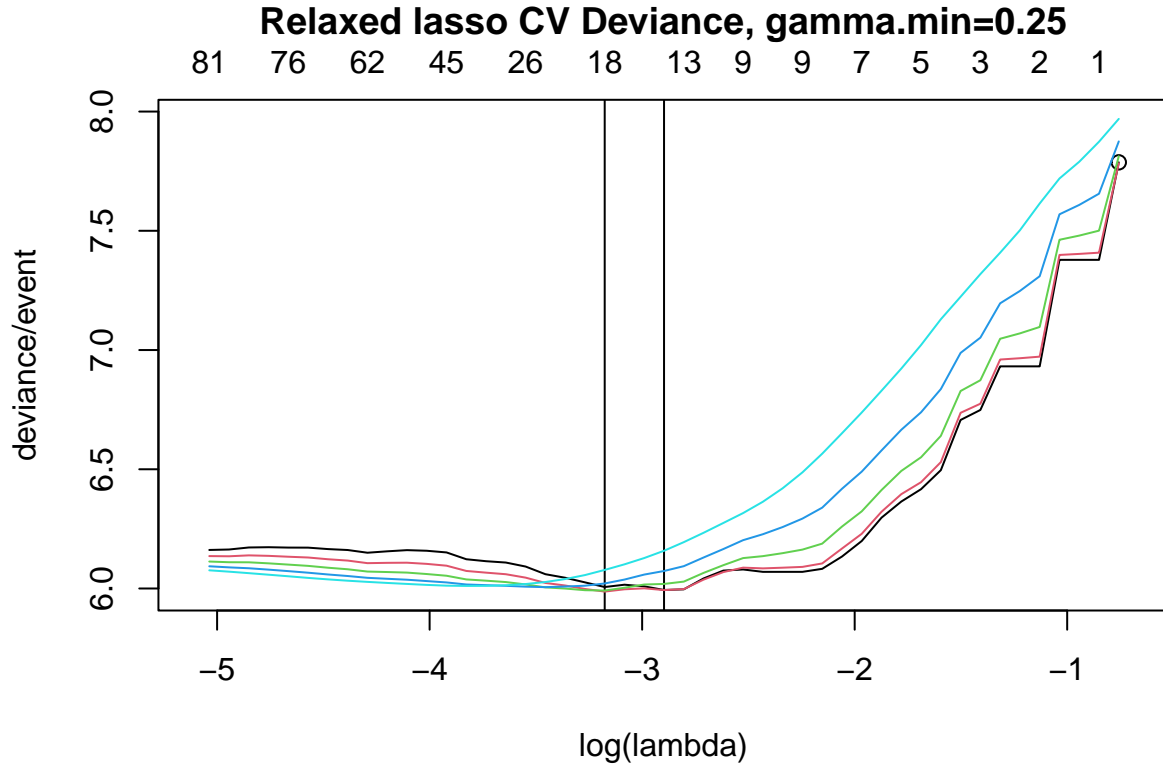
The performance measure estimates, here of concordance, from the individual (outer) cross validation for each fold are depicted by thin lines of different colors and styles, while the composite value from all folds is depicted by a thicker black line, and the performance estimates naively calculated 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 using the the full dataset, 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.glmnet()` 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 dataset.

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

```
## min CV average deviance (max log likelihood) for
##   relaxed at log(lambda) = -3.176, gamma.min = 0.25, df = 18
##   fully relaxed at log(lambda) = -2.897, df = 13
##   fully penalized at log(lambda) = -3.827, df = 42
```

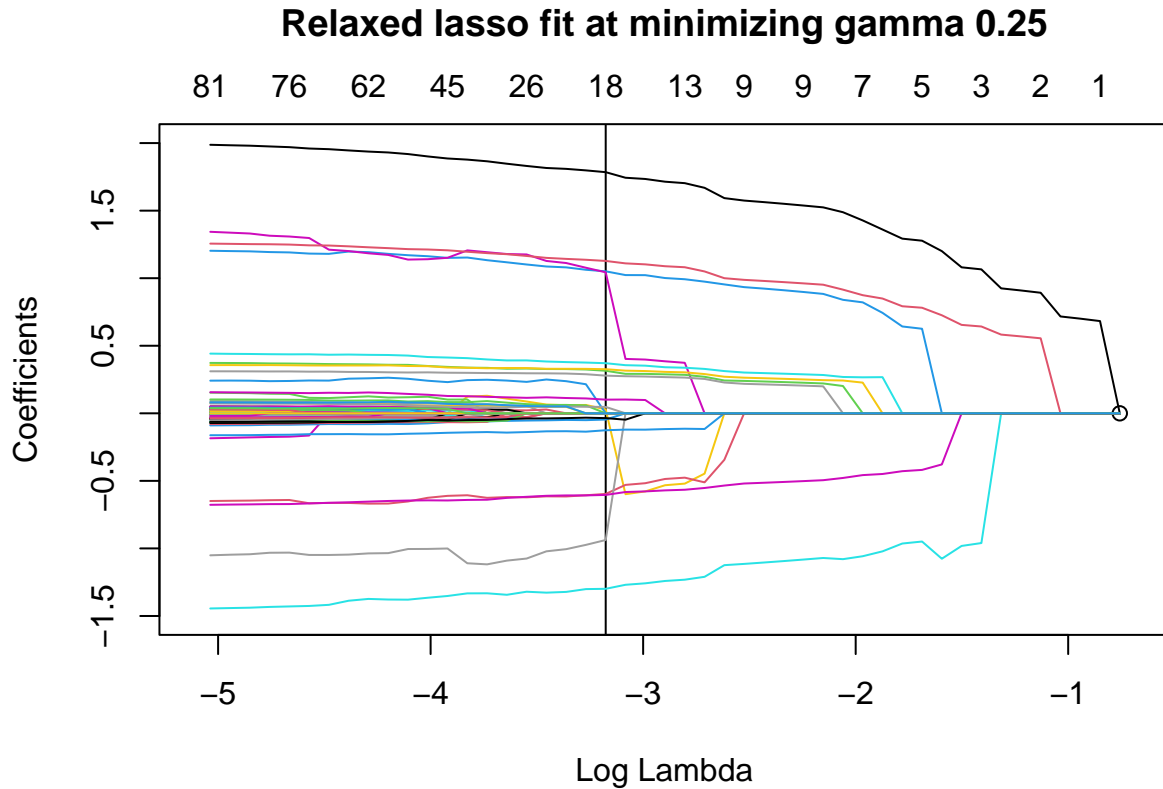


This figure has multiple lines, depicting deviance as a function of lambda for different gamma values. Whereas there is no legend here for gamma, when non-zero coefficients start to enter the model as the penalty is reduced, here shown at the right, deviances tend to be smaller for gamma = 0, greater for gamma = 1 and in between for other gammas values. The minimizing lambda and gamma pair is indicated by the left most 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. More relevant we see that the fully unpenalized lasso model fits (gamma=0) shown in a black line with a black circle at the largest lambda, achieves a minimal deviance at about log(lambda) = -2.9, and highlighted by the right most vertical line. The minimizing deviance for the fully relaxed lasso model is nearly that of the relaxed lasso model tuning for both lambda and gamma.

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

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

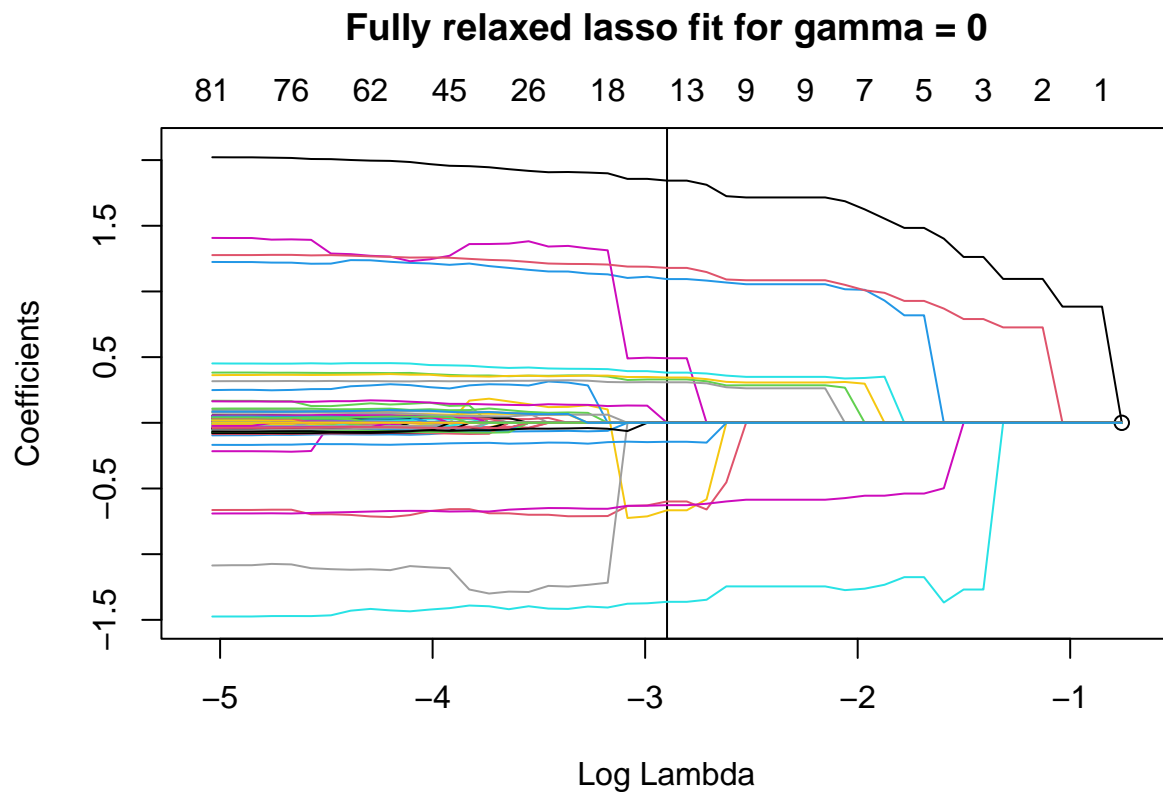
```
## min CV average deviance (max log likelihood)  
## at log(lambda.min) = -3.176, gamma.min = 0.25, df = 18
```



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.

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

```
## Fully relaxed min CV average deviance (max log likelihood)
## at log(lambda.min) = -2.897, df = 13
```



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, 13, to be substantially less than the 18 from the relaxed lasso fit and the 42 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.

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

## The relaxed minimum is obtained for lambda = 0.04174656 and gamma = 0.25
## with df (number of non-zero terms) = 18, average deviance = 5.987085 and beta =
##      X4      X5      X7      X10      X14
## 1.050952e+00 -1.298945e+00 2.912949e-02 -5.966936e-01 1.043615e+00
##      X15      X16      X18      X19      X20
## -1.719568e-16 -9.378331e-01 1.127645e+00 3.149391e-01 -1.253247e-01
##      X21      X22      X23      X24      X25
## 3.707304e-01 -6.039354e-01 3.263490e-01 2.789101e-01 1.784703e+00
##      X38      X60      X88      X97
## 1.011793e-01 -4.634282e-02 4.740086e-02 -3.554883e-02
##
## The fully relaxed (gamma=0) minimum is obtained at lambda = 0.0551865
## with df (number of non-zero terms) = 13, average deviance = 5.993744 and beta =
##      X4      X5      X7      X10      X14      X18      X19
## 1.0941352 -1.3627747 -0.6659735 -0.5988756 0.4916291 1.1792301 0.3302211
##      X20      X21      X22      X23      X24      X25
## -0.1444546 0.3820778 -0.6270995 0.3448224 0.3085746 1.8435742
##
## The UNrelaxed (gamma=1) minimum is obtained at lambda = 0.02176669
## with df (number of non-zero terms) = 42, average deviance = 6.010734
##
## Order coefficients entered into the lasso model (1st to last):
## [1] "X25" "X18" "X5" "X22" "X4" "X21" "X23" "X19" "X24" "X10"
## [11] "X7" "X20" "X14" "X38" "X97" "X16" "X60" "X88" "X12" "X43"
## [21] "X71" "X100" "X34" "X32" "X50" "X58" "X41" "X49" "X64" "X84"
## [31] "X91" "X98" "X39" "X40" "X66" "X73" "X74" "X11" "X61" "X69"
## [41] "X70" "X96"
```

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 predicted `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)
```

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

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

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

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

```
##          X1          X2          X3          X4          X5
## 0.000000e+00 0.000000e+00 0.000000e+00 1.050952e+00 -1.298945e+00
##          X6          X7          X8          X9          X10
## 0.000000e+00 2.912949e-02 0.000000e+00 0.000000e+00 -5.966936e-01
##          X11          X12          X13          X14          X15
## 0.000000e+00 0.000000e+00 0.000000e+00 1.043615e+00 -1.719568e-16
##          X16          X17          X18          X19          X20
## -9.378331e-01 0.000000e+00 1.127645e+00 3.149391e-01 -1.253247e-01
##          X21          X22          X23          X24          X25
## 3.707304e-01 -6.039354e-01 3.263490e-01 2.789101e-01 1.784703e+00
##          X26          X27          X28          X29          X30
## 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
##          X31          X32          X33          X34          X35
## 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
##          X36          X37          X38          X39          X40
## 0.000000e+00 0.000000e+00 1.011793e-01 0.000000e+00 0.000000e+00
##          X41          X42          X43          X44          X45
## 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
##          X46          X47          X48          X49          X50
## 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
##          X51          X52          X53          X54          X55
## 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
##          X56          X57          X58          X59          X60
## 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 -4.634282e-02
##          X61          X62          X63          X64          X65
## 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
##          X66          X67          X68          X69          X70
## 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
##          X71          X72          X73          X74          X75
```

```
## 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
##          X76          X77          X78          X79          X80
## 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
##          X81          X82          X83          X84          X85
## 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
##          X86          X87          X88          X89          X90
## 0.000000e+00 0.000000e+00 4.740086e-02 0.000000e+00 0.000000e+00
##          X91          X92          X93          X94          X95
## 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
##          X96          X97          X98          X99          X100
## 0.000000e+00 -3.554883e-02 0.000000e+00 0.000000e+00 0.000000e+00
```

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

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

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

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

## Nested cross validation (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 `glmnet.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 gaussian errors
# Use the same simulated data output object from above, that is from the call
# simdata=glmnet.simdata(nrows=1000, ncols=100, beta=NULL)
#
# extract linear regression model data
yg = simdata$y_ # outcome vector with Gaussian (normal) errors
# Get the ML model fits
nested.gau.fit = nested.glmnet(xs, NULL, yg, NULL, family="gaussian",
                             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)
```

```
# 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),
## 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
## LASSO min      0.8607 -0.0192  1.0184  0.8612
## LASSO minR     0.8594 -0.0148  1.0137  0.8596
## LASSO minR.GO  0.8570  0.0059  0.9899  0.8569
## Ridge         0.8492 -0.0822  1.0979  0.8568
##
## Ave Non Zero Naive DevRat Naive R-square Non Zero
## LASSO min      59.4      0.8794  0.9380  65
## LASSO minR     39.6      0.8794  0.9380  66
## LASSO minR.GO  27.8      0.8763  0.9361  25
## Ridge         99.0      0.8806  0.9400  99
##
##
## Ave DevRat Ave Int Ave Slope Ave R-square
## XGB (not tuned) 0.6739 -0.1104  1.1315  0.6864
## XGB lasso Feature 0.8108  0.0292  0.9957  0.8114
## XGB lasso Offset  0.8567  0.0043  0.9924  0.8568
## XGB Tuned       0.8119 -0.0432  1.0675  0.8158
## XGB Tuned lasso Feature 0.8510 -0.0358  1.0367  0.8527
## XGB Tuned lasso Offset 0.8595  0.0039  0.9925  0.8595
##
## Ave Non Zero Naive DevRat Naive R-square Non Zero
## XGB (not tuned) 100      0.9991  0.9992  100
## XGB lasso Feature 100      0.9976  0.9978  100
## XGB lasso Offset  100      0.8956  0.8957  100
## XGB Tuned       100      0.9138  0.9187  100
## XGB Tuned lasso Feature 100      0.8938  0.8962  100
## XGB Tuned lasso Offset 100      0.8799  0.8799  100
##
##
## Ave DevRat Ave Int Ave Slope Ave R-square
## RF Simple      0.7117 -0.2305  1.2175  0.7370
## RF lasso Feature 0.8470 -0.0563  1.0349  0.8482
## RF lasso Offset  0.8592  0.0066  0.9911  0.8594
##
## Ave Non Zero Naive DevRat Naive R-square Non Zero
## RF Simple      58.0      0.9285  0.9491  60
## RF lasso Feature 57.7      0.9630  0.9666  50
## RF lasso Offset  17.5      0.9636  0.9645  10
##
##
## Ave DevRat Ave Int Ave Slope Ave R-square
## ANN Uninformed  0.6879  0.1038  0.9464  0.6948
## ANN lasso terms, lasso feat 0.8312 -0.0152  0.9761  0.8326
## ANN lasso terms, wts reset 0.8586 -0.0461  0.9918  0.8595
##
## Ave Non Zero Naive DevRat Naive R-square Non Zero
## ANN Uninformed  100.0      0.9586  0.9588  100
## ANN lasso terms, lasso feat 39.6      0.9263  0.9266  66
## ANN lasso terms, wts reset 39.6      0.8723  0.8796  66

```

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. Note for the 2nd and 3rd ANN models, based upon the user specification of the *ensemble* option in the call to `nested.glmnetr()`, only those variables with non zero coefficients in the relaxed lasso model are given as input.

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

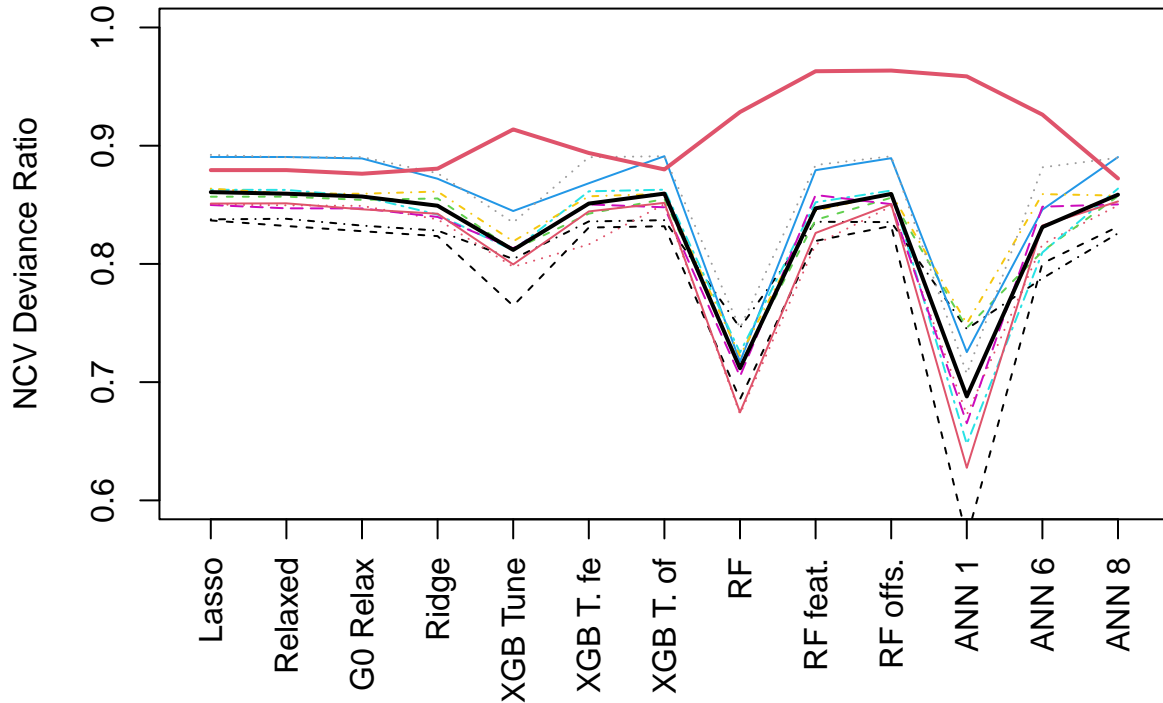
```
## put the data into a data frame
dframe = summary( nested.gau.fit , table = 0)
## use the roundperf() function from 'glmnetr' to round so long as it doesn't too
## obscure the original values
## print just the first 4 columns to get a more succinct table
roundperf(dframe, digits = 3)[,c(1:4)]
```

##	Ave DevRat	Ave Int	Ave Slope	Ave R-square
## LASSO min	0.861	-0.019	1.018	0.861
## LASSO minR	0.859	-0.015	1.014	0.860
## LASSO minR.GO	0.857	0.006	0.990	0.857
## Ridge	0.849	-0.082	1.098	0.857
## XGB (not tuned)	0.674	-0.110	1.131	0.686
## XGB lasso Feature	0.811	0.029	0.996	0.811
## XGB lasso Offset	0.857	0.004	0.992	0.857
## XGB Tuned	0.812	-0.043	1.068	0.816
## XGB Tuned lasso Feature	0.851	-0.036	1.037	0.853
## XGB Tuned lasso Offset	0.859	0.004	0.992	0.860
## RF Simple	0.712	-0.231	1.217	0.737
## RF lasso Feature	0.847	-0.056	1.035	0.848
## RF lasso Offset	0.859	0.007	0.991	0.859
## ANN Uninformed	0.688	0.104	0.946	0.695
## ANN lasso terms, lasso feat	0.831	-0.015	0.976	0.833
## ANN lasso terms, wts reset	0.859	-0.046	0.992	0.859

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))
```

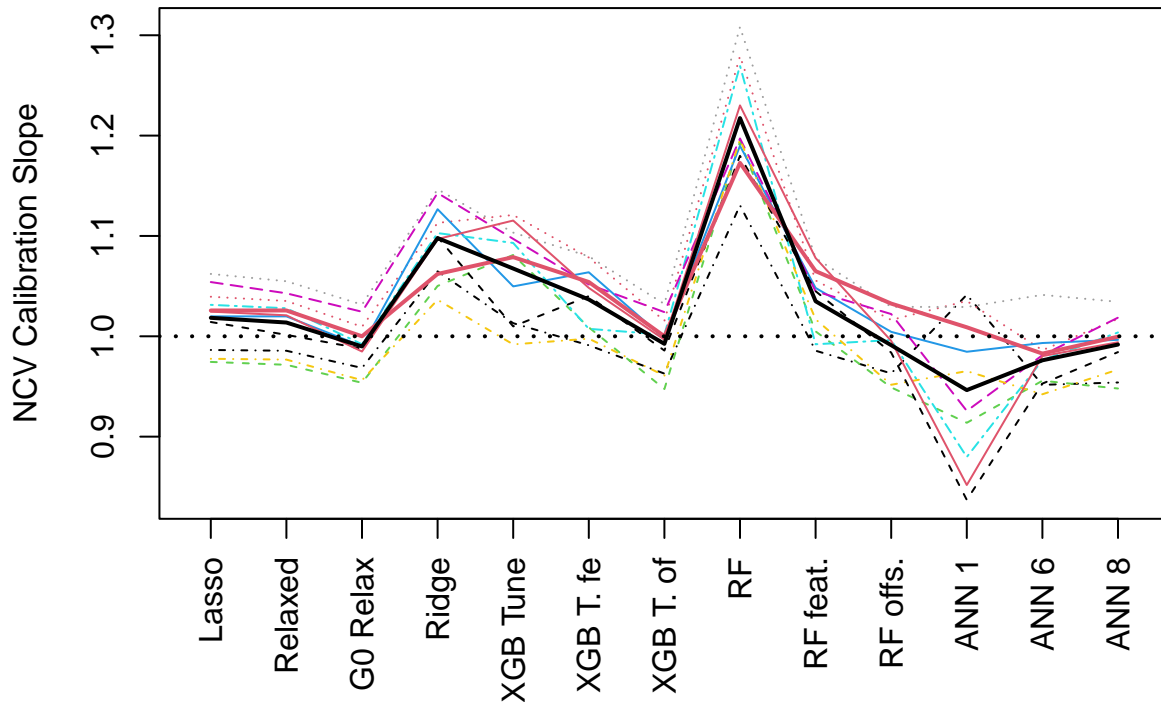


For “gaussian” data, i.e generalization of the linear regression model, the deviance ratio is the reduction Mean Square Error (MSE) achieved by the machine learning model relative to the MSE of the model based upon only the overall average.

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  $X \cdot \beta$  predicted, 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.glmnet()` output object with names like `cv_glmnet_fit`, `xgb.simple.fit`, `xgb.tuned.fit`, `rf.tuned.fit`, `cv.stepreg.fit` and `ann_fit_X` with X corresponding to the ensemble designation. `cv_glmnet_fit` has a similar yet different format to that of `cv.glmnet()` by including further fit information. The XGB outputs are 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 hyperparameters using the ‘mlrMBO’ 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 datasets like `xx.devian.cv`, `xx.lincal.cv`, `xx.agree.cv` for further processing by the `summary()` function or by the user. For example

```
# Manually calculate CV R_square for lasso models
corr.cv = nested.gau.fit$lasso.agree.cv
avecorr = colMeans(corr.cv)
R_square = round( avecorr ^2 , digits = 4)
R_square
```

```
##      1se      min      1seR      minR 1seR.GO minR.GO      ridge
## 0.8588 0.8612 0.8593 0.8596 0.8602 0.8569 0.8568
```

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

## Further model assessment

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

```
# Get predicted values from CV relaxed lasso model embedded in nested CV outputs & Plot
xb.hat = predict(object=nested.cox.fit, xs_new=xs, lam=NULL, gam=NULL, comment=FALSE)
# describe the distribution of xb.hat
round(1000*quantile(xb.hat, c(0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 0.90, 0.95, 0.99)))/1000
```

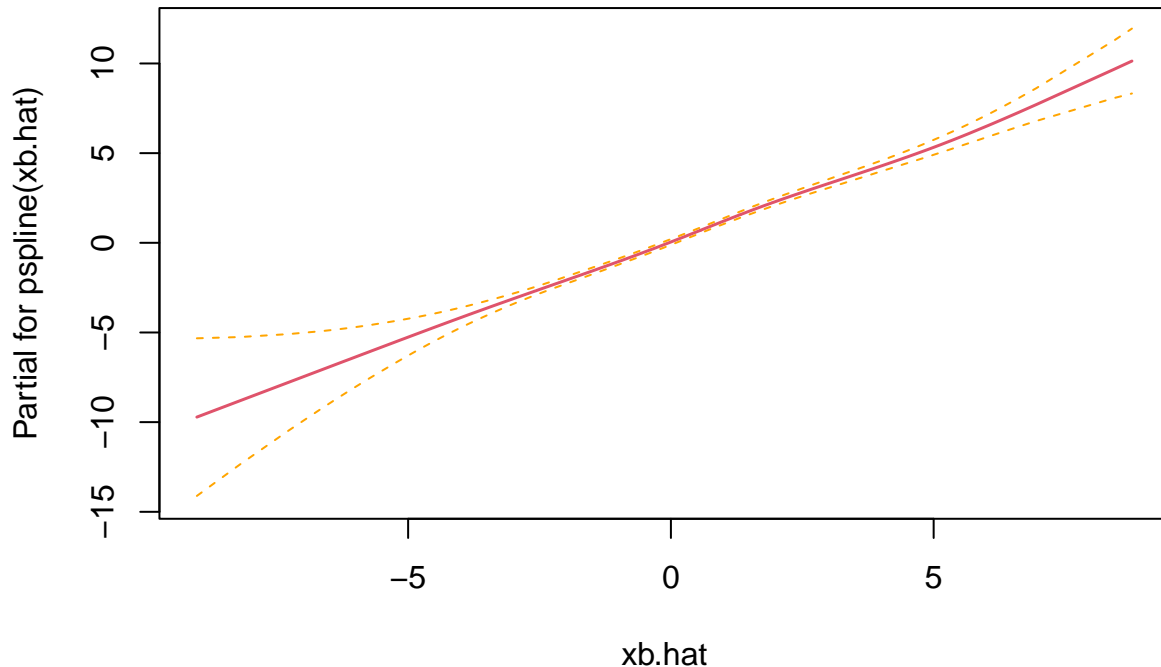
```
##      1%      5%     10%    25%    50%    75%    90%    95%    99%
## -5.839 -4.122 -3.233 -1.804 -0.070  1.578  3.188  3.989  5.449
```

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

```
## Call:
## coxph(formula = Surv(y_, 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 predicted.

## Model replicability and model comparisons

To facilitate reproducible results the `nested.glmnet()` 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.glmnet()`, one can reproduce the results from this earlier call. Because the seeds are saved for folds both in the inner and outer loop, 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.glmnet()`, 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 unwantingly 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.glmnet()` 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 Y & Grandvalet Y, "No Unbiased Estimator of the Variance of K-Fold

Cross-Validation”, Journal of Machine Learning Research 5 (2004) 1089–1105). Still, people use these SDs routinely as a rough approximation, being cautious of making strong inferences. We expect any dependencies at least between the paired differences should be minimal and reasonable in approximation. This intuition is supported by informal simulations. The reader is invited to perform similar studies using the type of data they encounter.

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
glmnetr.compcv(nested.gau.fit, type="devrat")

## Ensemble option used when fitting models : (1,0,0,0, 0,1,0,1)
##
## Model performance comparison in terms of ** Deviance Ratio **
##
## Comparison                estimate    (95% CI)      p
##
## lasso.minR - lasso.min      -0.0014 (-0.0027, 0)  0.0489
## lasso.minR - lasso.minR0    0.0025 (8e-04, 0.0043)  0.0096
## lasso.min - lasso.minR0     0.0039 (0.0021, 0.0057)  7e-04
##
## XGBoost (tuned) - XGBoost (simple)  0.1363 (0.0926, 0.1799)  1e-04
## XGBoost (tuned) lasso feature - no feature  0.0396 (0.0292, 0.0501)  0
## XGBoost (tuned) lasso offset - no offset  0.0477 (0.0402, 0.0552)  0
##
## RF with lasso feature - no feature  0.1344 (0.1197, 0.1492)  0
## RF with lasso offset - no offset    0.1467 (0.1285, 0.1649)  0
##
## ANN with with lasso feature - no feature  0.1439 (0.1009, 0.1869)  0
## ANN with with lasso offset - no offset    0.1715 (0.1306, 0.2124)  0
##
## lasso.minR - XGB (tuned)          0.0477 (0.0404, 0.0549)  0
## lasso.minR - XGB with lasso feature  0.008 (-3e-04, 0.0164)  0.0571
## lasso.minR - XGB with lasso offset    0 (-7e-04, 7e-04)  0.8843
## lasso.minR - Random Forest          0.1469 (0.1289, 0.1649)  0
## lasso.minR - RF with lasso feature    0.0125 (0.0037, 0.0212)  0.0105
## lasso.minR - RF with lasso offset     0.0027 (0.001, 0.0044)  0.0055
## lasso.minR - ANN                    0.1724 (0.1332, 0.2115)  0
## lasso.minR - ANN l lasso feature      0.0285 (0.0137, 0.0433)  0.0018
## lasso.minR - ANN l lasso offset (upated)  9e-04 (-0.0022, 0.0039)  0.5275
##
## XGBoost (tuned) - RF                0.0993 (0.0832, 0.1154)  0
## XGBoost lasso feature-RF with lasso feature  0.0044 (-0.0021, 0.011)  0.1607
## XGBoost lasso offset- RF with lasso offset  3e-04 (-8e-04, 0.0013)  0.5565
## XGBoost (tuned) - ANN                0.1247 (0.0908, 0.1586)  0
## XGBoost lasso feature - ANN, l lasso feature  0.0205 (0.0065, 0.0345)  0.0091
## XGBoost lasso offset-ANN l lasso offset(upated)  9e-04 (-0.0019, 0.0037)  0.4683
##
## RF - ANN                            0.0255 (-0.0085, 0.0594)  0.1241
## RF lasso feature - ANN l lasso feature  0.016 (9e-04, 0.0312)  0.0404
## RF lasso offset - ANN l lasso offset (upated)  7e-04 (-0.0018, 0.0031)  0.5593
```

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