A Tutorial on Building Nomograms for Penalized Cox Models with High-Dimensional Data

Nan Xiao, Qing-Song Xu, Miao-Zhu Li

hdnom version 4.0

July 25, 2016

hdnom.org

Contents

1 Introduction **1**

2 Build Survival Models **1**

3 Nomogram Plotting **2**

4 Model Validation **4**
4.1 Internal Validation 4
4.2 External Validation 6

5 Model Calibration **7**
5.1 Internal Calibration 7
5.2 External Calibration 8
5.3 Kaplan-Meier Analysis for Risk Groups 10
5.4 Log-Rank Test for Risk Groups 11

6 Model Comparison **13**
6.1 Model Comparison by Validation 14
6.2 Model Comparison by Calibration 15

7 Prediction on New Data **18**

8 Customize Color Palette **19**

References **19**

1 Introduction

It is a challenging task to model the emerging high-dimensional clinical data with survival outcomes. For its simplicity and efficiency, penalized Cox models are significantly useful for accomplishing such tasks.

`hdnom` streamlines the workflow of high-dimensional Cox model building, nomogram plotting, model validation, calibration, and comparison. To load the package in R, simply type:

```
library("hdnom")
```

2 Build Survival Models

To build a penalized Cox model with good predictive performance, some parameter tuning is usually needed. For example, the elastic-net model requires to tune the ℓ_1-ℓ_2 penalty trade-off parameter α, and the regularization parameter λ.

To free the users from the tedious and error-prone parameter tuning process, `hdnom` provides several functions for automatic parameter tuning and model selection, including the following model types:

Function Name	Model Type
`hdcox.lasso()`	Lasso
`hdcox.alasso()`	Adaptive lasso
`hdcox.flasso()`	Fused lasso
`hdcox.enet()`	Elastic-net
`hdcox.aenet()`	Adaptive elastic-net
`hdcox.mcp()`	MCP
`hdcox.mnet()`	Mnet (MCP + elastic-net)
`hdcox.scad()`	SCAD
`hdcox.snet()`	Snet (SCAD + elastic-net)

In the next, we will use the imputed SMART study data (Steyerberg 2008) to demonstrate a complete process of model building, nomogram plotting, model validation, calibration, and comparison with `hdnom`.

Load the `smart` dataset:

```
data("smart")
x = as.matrix(smart[, -c(1, 2)])
time = smart$TEVENT
event = smart$EVENT

library("survival")
y = Surv(time, event)
```

The dataset contains 3,873 observations with corresponding survival outcome (`time`, `event`). 27

clinical variables (`x`) are available as the predictors. See `?smart` for a detailed explanation of the variables.

Fit a penalized Cox model by adaptive elastic-net regularization, with `hdcox.aenet`:

```
# Enable parallel parameter tuning
suppressMessages(library("doParallel"))
registerDoParallel(detectCores())

aenetfit = hdcox.aenet(x, y, nfolds = 10, rule = "lambda.1se",
                       seed = c(5, 7), parallel = TRUE)
names(aenetfit)
```

```
## [1] "seed"              "enet_best_alpha"   "enet_best_lambda"
## [4] "enet_model"        "aenet_best_alpha"  "aenet_best_lambda"
## [7] "aenet_model"       "pen_factor"
```

Adaptive elastic-net includes two estimation steps. The random seed used for parameter tuning, the selected best α, the selected best λ, the model fitted for each estimation step, and the penalty factor for the model coefficients in the second estimation step are all stored in the model object `aenetfit`.

3 Nomogram Plotting

Before plotting the nomogram, we need to extract some necessary information about the model, namely, the model object and parameters, from the result of the last step:

```
fit    = aenetfit$aenet_model
alpha  = aenetfit$aenet_best_alpha
lambda = aenetfit$aenet_best_lambda
adapen = aenetfit$pen_factor
```

To plot the nomogram, first we make `x` available as a `datadist` object for the `rms` package (Harrell 2015), then generate a `hdnom.nomogram` object with `hdnom.nomogram()`, and plot the nomogram:

```
suppressMessages(library("rms"))
x.df = as.data.frame(x)
dd = datadist(x.df)
options(datadist = "dd")

nom = hdnom.nomogram(fit, model.type = "aenet", x, time, event, x.df,
                     lambda = lambda, pred.at = 365 * 2,
                     funlabel = "2-Year Overall Survival Probability")
plot(nom)
```

According to the nomogram, the Cox model with adaptive elastic-net penalty selected six variables (`AGE`, `AAA`, `STENOSIS`, `HDL`, `IMT`, `ALBUMIN`) from the original set of 27 variables, effectively reduced model complexity.

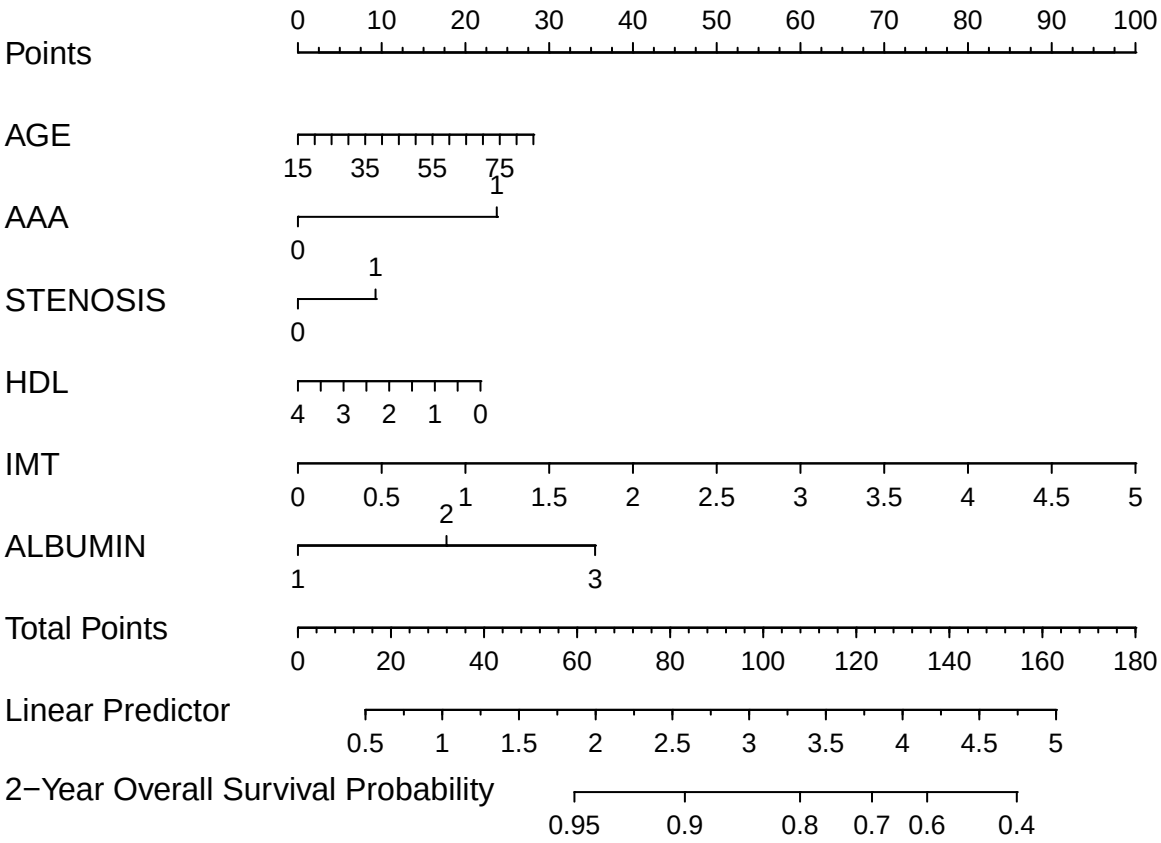

Figure 1: Nomogram for the adaptive elastic-net model

Information about the nomogram itself, such as the point-linear predictor unit mapping and total points-survival probability mapping, can be viewed by printing the `nom` object directly.

4 Model Validation

It is a common practice to utilize resampling-based methods to validate the predictive performance of a penalized Cox model. Bootstrap, k-fold cross-validation, and repeated k-fold cross-validation are the most employed methods for such purpose.

`hdnom` supports both internal model validation and external model validation. Internal validation takes the dataset used to build the model and evaluates the predictive performance on the data internally with the above resampling-based methods, while external validation evaluates the model's predictive performance on a dataset which is independent to the dataset used to build the model.

4.1 Internal Validation

`hdnom.validate()` allows us to assess the model performance internally by time-dependent AUC (Area Under the ROC Curve) with the above three resampling methods.

Here, we validate the performance of the adaptive elastic-net model with bootstrap resampling, at every half year from the second year to the fifth year:

```
val.int = hdnom.validate(x, time, event, model.type = "aenet",
                         alpha = alpha, lambda = lambda, pen.factor = adapen,
                         method = "bootstrap", boot.times = 200,
                         tauc.type = "UNO", tauc.time = seq(2, 5, 0.5) * 365,
                         seed = 42, trace = FALSE)
print(val.int)
```

```
## High-Dimensional Cox Model Validation Object
## Random seed: 42
## Validation method: bootstrap
## Bootstrap samples: 200
## Model type: aenet
## glmnet model alpha: 0.15
## glmnet model lambda: 2.559313e+12
## glmnet model penalty factor: specified
## Time-dependent AUC type: UNO
## Evaluation time points for tAUC: 730 912.5 1095 1277.5 1460 1642.5 1825
```

```
summary(val.int)
```

```
## Time-Dependent AUC Summary at Evaluation Time Points
```

```
##                730     912.5      1095    1277.5      1460    1642.5
## Mean     0.6838358 0.6774956 0.7104212 0.7313414 0.6717074 0.6727783
## Min      0.6690070 0.6672240 0.6991432 0.7202336 0.6572806 0.6608793
```

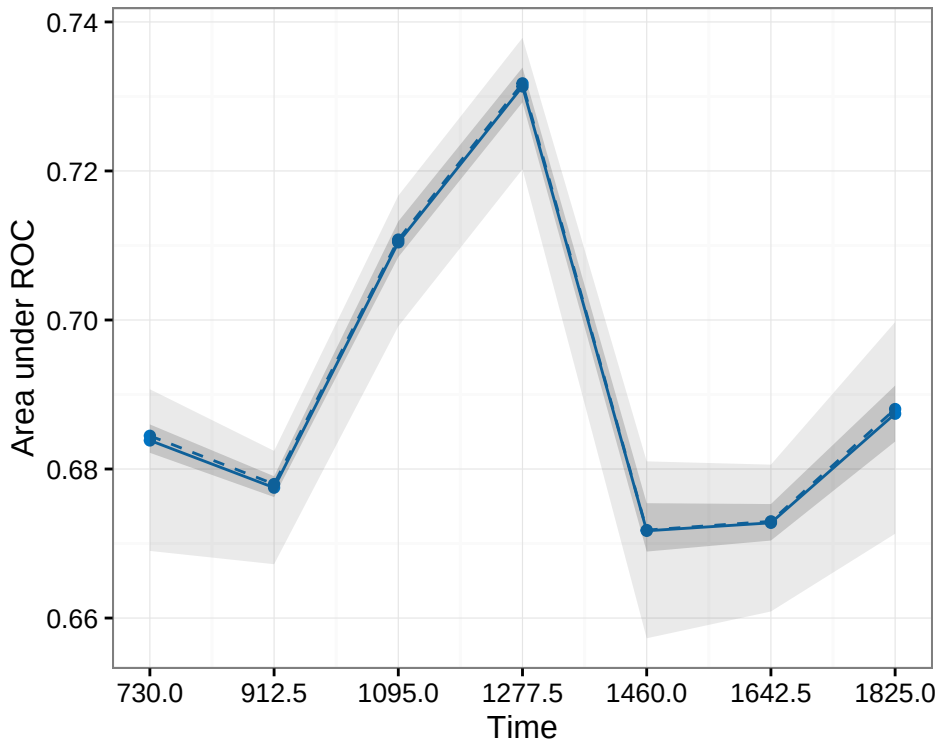

Figure 2: Time-dependent AUC values for internal model validation

```
## 0.25 Qt. 0.6821762 0.6762373 0.7084481 0.7292107 0.6689195 0.6703975
## Median   0.6844577 0.6779749 0.7107909 0.7317549 0.6718038 0.6729762
## 0.75 Qt. 0.6859822 0.6790314 0.7132373 0.7338841 0.6754134 0.6752983
## Max      0.6907049 0.6824337 0.7166890 0.7378667 0.6810270 0.6805885
##                1825
## Mean     0.6874231
## Min      0.6713106
## 0.25 Qt. 0.6836848
## Median   0.6880464
## 0.75 Qt. 0.6911997
## Max      0.6997200
```

The mean, median, 25%, and 75% quantiles of time-dependent AUC at each time point across all bootstrap predictions are listed above. The median and the mean can be considered as the bias-corrected estimation of the model performance.

It is also possible to plot the model validation result:

```
plot(val.int)
```

The solid line represents the mean of the AUC, the dashed line represents the median of the AUC. The darker interval in the plot shows the 25% and 75% quantiles of AUC, the lighter interval shows the minimum and maximum of AUC.

It seems that the bootstrap-based validation result is stable: the median and the mean value at

each evaluation time point are close; the 25% and 75% quantiles are also close to the median at each time point. An argument `ylim` in the plot functions for such model validation objects can be set to manually adjust the range of `y` coordinates.

Bootstrap-based validation often gives relatively stable results. Many of the established nomograms in clinical oncology research are validated by bootstrap methods. K-fold cross-validation provides a more strict evaluation scheme than bootstrap. Repeated cross-validation gives similar results as k-fold cross-validation, and usually more robust. These two methods are more applied by the machine learning community. Check `?hdnom.validate` for more examples about internal model validation.

4.2 External Validation

Now we have the internally validated model. To perform external validation, we usually need an independent dataset (preferably, collected in other studies), which has the same variables as the dataset used to build the model. For penalized Cox models, the external dataset should have at least the same variables that have been selected in the model.

For demonstration purposes, here we draw 1,000 samples from the `smart` data and *assume* that they form an external validation dataset, then use `hdnom.external.validate()` to perform external validation:

```
x_new = as.matrix(smart[, -c(1, 2)])[1001:2000, ]
time_new = smart$TEVENT[1001:2000]
event_new = smart$EVENT[1001:2000]

# External validation with time-dependent AUC
val.ext =
  hdnom.external.validate(aenetfit, x, time, event,
                          x_new, time_new, event_new,
                          tauc.type = "UNO",
                          tauc.time = seq(0.5, 2, 0.25) * 365)

print(val.ext)
```

```
## High-Dimensional Cox Model External Validation Object
## Model type: aenet
## Time-dependent AUC type: UNO
## Evaluation time points for tAUC: 182.5 273.75 365 456.25 547.5 638.75 730
```

```
summary(val.ext)
```

```
## Time-Dependent AUC Summary at Evaluation Time Points
```

```
##         182.5    273.75       365    456.25     547.5    638.75       730
## AUC 0.5608442 0.6299909 0.6289888 0.6530596 0.6728417 0.6795879 0.6922393
```

```
plot(val.ext, ylim = c(0.5, 0.8))
```

The time-dependent AUC on the external dataset is shown above.

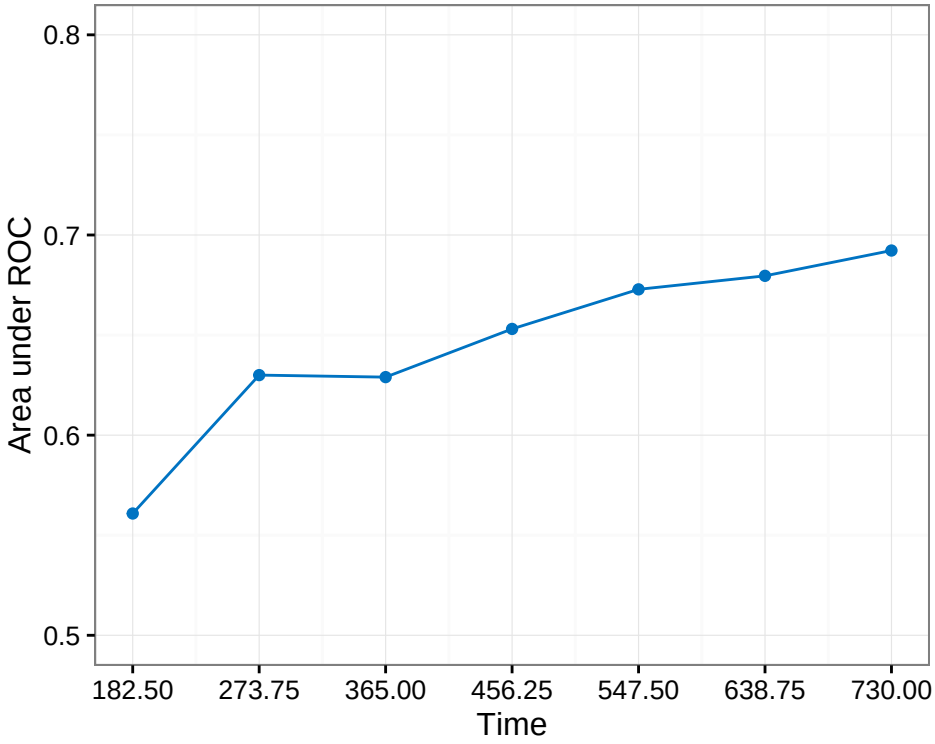

Figure 3: Time-dependent AUC values for external model validation

5 Model Calibration

Measuring how far the model predictions are from actual survival outcomes is known as *calibration*. Calibration can be assessed by plotting the predicted probabilities from the model versus actual survival probabilities. Similar to model validation, both internal model calibration and external model calibration are supported in `hdnom`.

5.1 Internal Calibration

`hdnom.calibrate()` provides non-resampling and resampling methods for internal model calibration, including direct fitting, bootstrap resampling, k-fold cross-validation, and repeated cross-validation.

For example, to calibrate the model internally with the bootstrap method:

```
cal.int = hdnom.calibrate(x, time, event, model.type = "aenet",
                          alpha = alpha, lambda = lambda, pen.factor = adapen,
                          method = "bootstrap", boot.times = 200,
                          pred.at = 365 * 5, ngroup = 3,
                          seed = 42, trace = FALSE)
print(cal.int)
```

```
## High-Dimensional Cox Model Calibration Object
```

```
## Random seed: 42
## Calibration method: bootstrap
## Bootstrap samples: 200
## Model type: aenet
## glmnet model alpha: 0.15
## glmnet model lambda: 2.559313e+12
## glmnet model penalty factor: specified
## Calibration time point: 1825
## Number of groups formed for calibration: 3
```

```
summary(cal.int)
```

```
##   Calibration Summary Table
##   Predicted  Observed Lower 95% Upper 95%
## 1 0.8032792 0.7584323 0.7297862 0.7882027
## 2 0.8933650 0.8939150 0.8726762 0.9156706
## 3 0.9225957 0.9334446 0.9158670 0.9513596
```

Here we splitted the samples into three risk groups. In practice, the number of risk groups is decided by the users according to their needs.

The model calibration results (the median of the predicted survival probability; the median of the observed survival probability estimated by Kaplan-Meier method with 95% CI) are summarized as above.

Plot the (internal) model calibration results:

```
plot(cal.int, xlim = c(0.5, 1), ylim = c(0.5, 1))
```

In practice, you may want to perform model calibration for multiple time points separately, and put the plots together in one figure. See `?hdnom.calibrate` for more examples about internal model calibration.

5.2 External Calibration

To perform external calibration with an external dataset, use `hdnom.external.calibrate()`:

```
cal.ext =
  hdnom.external.calibrate(aenetfit, x, time, event,
                           x_new, time_new, event_new,
                           pred.at = 365 * 5, ngroup = 3)
```

```
print(cal.ext)
```

```
## High-Dimensional Cox Model External Calibration Object
## Model type: aenet
## Calibration time point: 1825
## Number of groups formed for calibration: 3
```

```
summary(cal.ext)
```

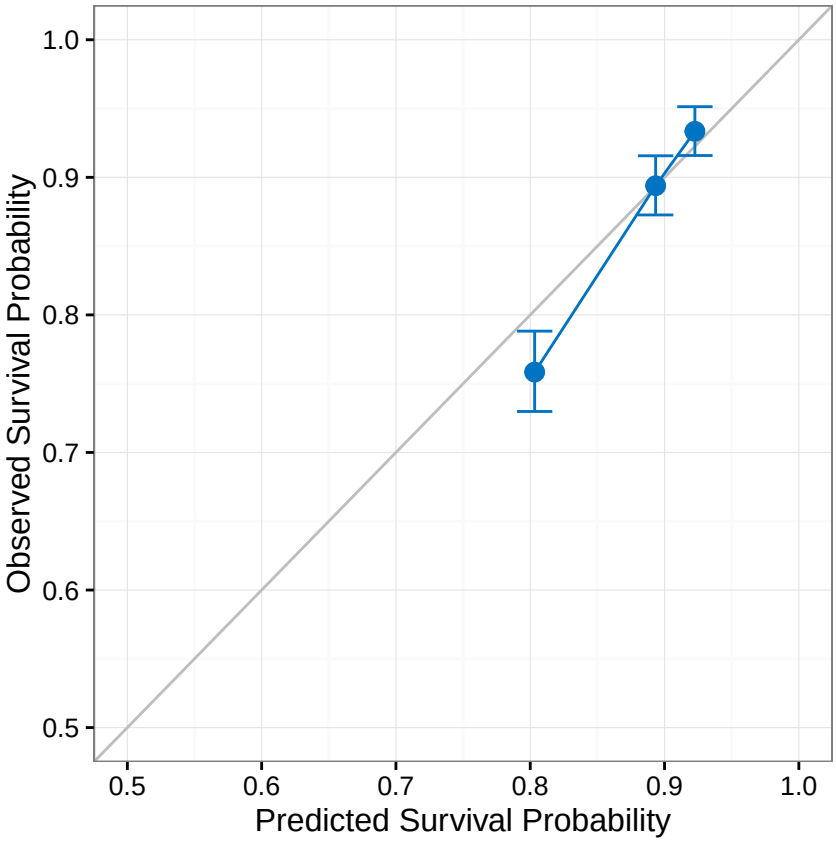

Figure 4: Internal model calibration plot

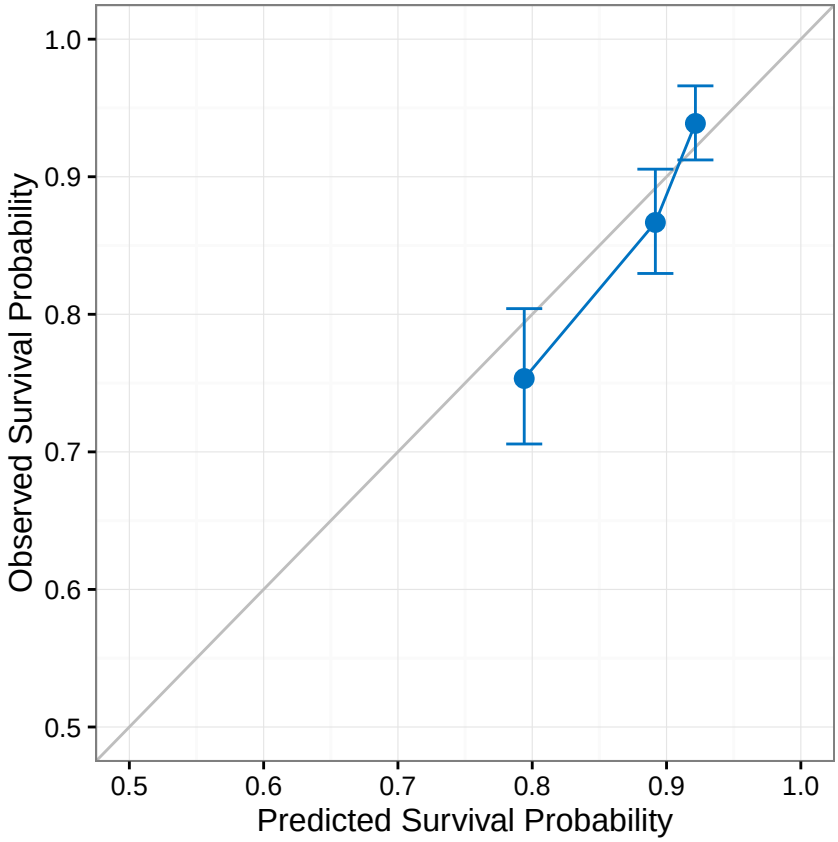

Figure 5: External model calibration plot

```
##   External Calibration Summary Table
##   Predicted  Observed Lower 95% Upper 95%
## 1 0.7940258 0.7533312 0.7057400 0.8041316
## 2 0.8916822 0.8667762 0.8296887 0.9055215
## 3 0.9214927 0.9387588 0.9122184 0.9660715
```

```
plot(cal.ext, xlim = c(0.5, 1), ylim = c(0.5, 1))
```

The external calibration results have the similar interpretations as the internal calibration results, except the fact that external calibration is performed on the external dataset.

5.3 Kaplan-Meier Analysis for Risk Groups

Internal calibration and external calibration both classify the testing set into different risk groups. For internal calibration, the testing set means all the samples in the dataset that was used to build the model, for external calibration, the testing set means the samples from the external dataset.

We can further analyze the differences in survival time for different risk groups with Kaplan-Meier survival curves and a number at risk table. For example, here we plot the Kaplan-Meier survival

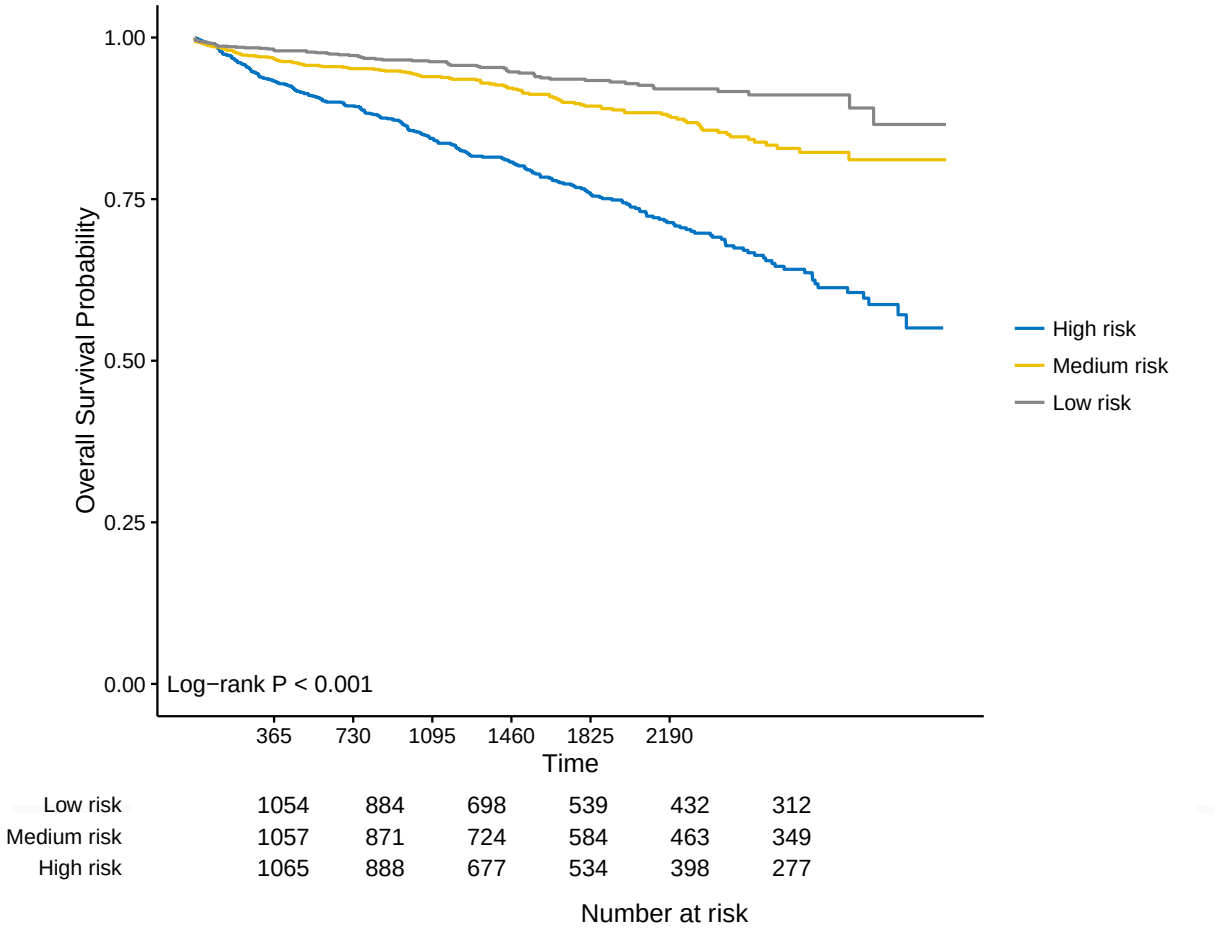

Figure 6: Kaplan-Meier survival curves for the three risk groups (internal calibration)

curves and evaluate the number at risk from one year to six years for the three risk groups, with the function `hdnom.kmplot()`:

```
hdnom.kmplot(cal.int, group.name = c('High risk', 'Medium risk', 'Low risk'),
             time.at = 1:6 * 365)

hdnom.kmplot(cal.ext, group.name = c('High risk', 'Medium risk', 'Low risk'),
             time.at = 1:6 * 365)
```

The p-value of the log-rank test is also shown in the plot.

5.4 Log-Rank Test for Risk Groups

To compare the differences between the survival curves (of the risk groups), log-rank test is often applied. `hdnom.logrank()` performs such tests on the internal calibration and external calibration results:

```
cal.int.logrank = hdnom.logrank(cal.int)
cal.int.logrank
```

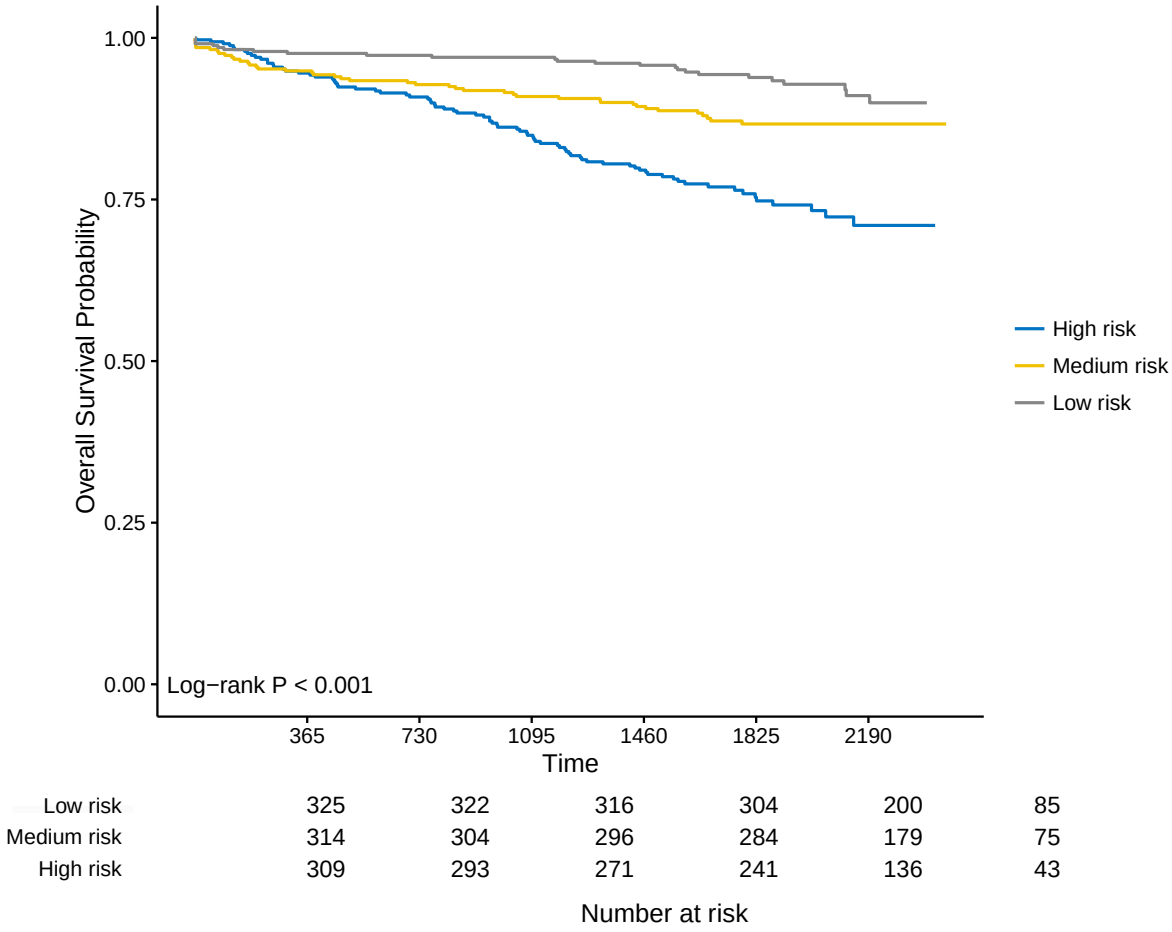

Figure 7: Kaplan-Meier survival curves for the three risk groups (external calibration)

```
## Call:
## survdiff(formula = formula("Surv(time, event) ~ grp"))
##
## n=3872, 1 observation deleted due to missingness.
##
##          N Observed Expected (O-E)^2/E (O-E)^2/V
## grp=1 1290      276      151     103.3     154.1
## grp=2 1291      118      158      10.1      15.4
## grp=3 1291       66      151      47.8      71.3
##
##  Chisq= 162  on 2 degrees of freedom, p= 0
```

```
cal.int.logrank$pval
```

```
## [1] 8.576116e-36
```

```
cal.ext.logrank = hdnom.logrank(cal.ext)
cal.ext.logrank
```

```
## Call:
## survdiff(formula = formula("Surv(time, event) ~ grp"))
##
## n=999, 1 observation deleted due to missingness.
##
##         N Observed Expected (O-E)^2/E (O-E)^2/V
## grp=1 333       81     45.2     28.37     41.11
## grp=2 333       42     49.5      1.15      1.74
## grp=3 333       24     52.3     15.28     23.75
##
##  Chisq= 45  on 2 degrees of freedom, p= 1.72e-10
```

```
cal.ext.logrank$pval
```

```
## [1] 1.720597e-10
```

The exact p-values for log-rank tests are stored as `cal.int.logrank$pval` and `cal.ext.logrank$pval`. Here $p < 0.001$ indicates significant differences between the survival curves for different risk groups.

6 Model Comparison

Given all the available model types, it might be a natural question for us to ask: which type of model performs the best for my data? Such model type selection problems can be (partially) solved by the built-in model comparison functions in `hdnom`.

6.1 Model Comparison by Validation

We can compare the model performance using time-dependent AUC by the same (internal) model validation approach as before. For example, here we compare lasso and elastic-net by repeated cross-validation (10 fold, repeat 10 times):

```
cmp.val =
  hdnom.compare.validate(x, time, event,
                         model.type = c("lasso", "enet"),
                         method = "repeated.cv", nfolds = 10, rep.times = 10,
                         tauc.type = "UNO",
                         tauc.time = seq(0.5, 2, 0.25) * 365,
                         seed = 42, trace = FALSE)

print(cmp.val)
```

```
## High-Dimensional Cox Model Validation Object
## Random seed: 42
## Validation method: repeated cross-validation
## Cross-validation folds: 10
## Cross-validation repeated times: 10
## Model type: lasso
## glmnet model alpha: 1
## glmnet model lambda: 0.01952497
## glmnet model penalty factor: not specified
## Time-dependent AUC type: UNO
## Evaluation time points for tAUC: 182.5 273.75 365 456.25 547.5 638.75 730
##
## High-Dimensional Cox Model Validation Object
## Random seed: 42
## Validation method: repeated cross-validation
## Cross-validation folds: 10
## Cross-validation repeated times: 10
## Model type: enet
## glmnet model alpha: 0.1
## glmnet model lambda: 0.1117291
## glmnet model penalty factor: not specified
## Time-dependent AUC type: UNO
## Evaluation time points for tAUC: 182.5 273.75 365 456.25 547.5 638.75 730
```

```
summary(cmp.val)
```

```
## Model type: lasso
##                       182.5    273.75       365    456.25     547.5
## Mean of Mean      0.6279719 0.6634524 0.6619716 0.6680223 0.6860378
## Mean of Min       0.4858318 0.5472671 0.5311816 0.5387785 0.5560724
## Mean of 0.25 Qt.  0.5705519 0.6109838 0.6234290 0.6249972 0.6487304
## Mean of Median    0.6214917 0.6586000 0.6592762 0.6699712 0.6877898
```

```
## Mean of 0.75 Qt. 0.6676840 0.7017052 0.7043178 0.7124661 0.7285051
## Mean of Max      0.8097248 0.8129838 0.7980135 0.7983564 0.8007999
##                      638.75       730
## Mean of Mean     0.6830995 0.6757933
## Mean of Min      0.5749002 0.5475929
## Mean of 0.25 Qt. 0.6413609 0.6317078
## Mean of Median   0.6839922 0.6811731
## Mean of 0.75 Qt. 0.7235427 0.7208864
## Mean of Max      0.7844050 0.7928456
##
## Model type: enet
##                       182.5    273.75       365    456.25     547.5
## Mean of Mean     0.6284894 0.6666666 0.6638599 0.6731489 0.6889693
## Mean of Min      0.4865484 0.5509019 0.5354278 0.5443231 0.5560948
## Mean of 0.25 Qt. 0.5742445 0.6148498 0.6215596 0.6296455 0.6497017
## Mean of Median   0.6171197 0.6604381 0.6566286 0.6718296 0.6901206
## Mean of 0.75 Qt. 0.6726565 0.7007647 0.7078837 0.7190586 0.7355692
## Mean of Max      0.8048343 0.8160801 0.8055489 0.8065498 0.8045257
##                      638.75       730
## Mean of Mean     0.6884454 0.6822229
## Mean of Min      0.5766107 0.5530374
## Mean of 0.25 Qt. 0.6515440 0.6400564
## Mean of Median   0.6906430 0.6888784
## Mean of 0.75 Qt. 0.7277501 0.7244514
## Mean of Max      0.7889265 0.8011706
```

```
plot(cmp.val, ylim = c(0.55, 0.75))
```

The solid line, dashed line and intervals have the same interpretation as above. For this comparison, there seems to be no substantial difference (AUC difference $< 5\%$) between lasso and elastic-net in terms of predictive performance, although elastic-net performs slightly better than lasso, at least for the last four time points in comparison.

The model comparison functions in `hdnom` have a minimal input design so you do not have to set the parameters for each model type manually. The program will try to determine the best parameter settings automatically for each model type to achieve the best performance.

6.2 Model Comparison by Calibration

It is also possible to compare the models by comparing their (internal) model calibration performance. To continue the example, we split the samples into five risk groups, and compare lasso to elastic-net via calibration:

```
cmp.cal =
  hdnom.compare.calibrate(x, time, event,
                          model.type = c("lasso", "enet"),
                          method = "repeated.cv", nfolds = 10, rep.times = 10,
                          pred.at = 365 * 9, ngroup = 5,
```

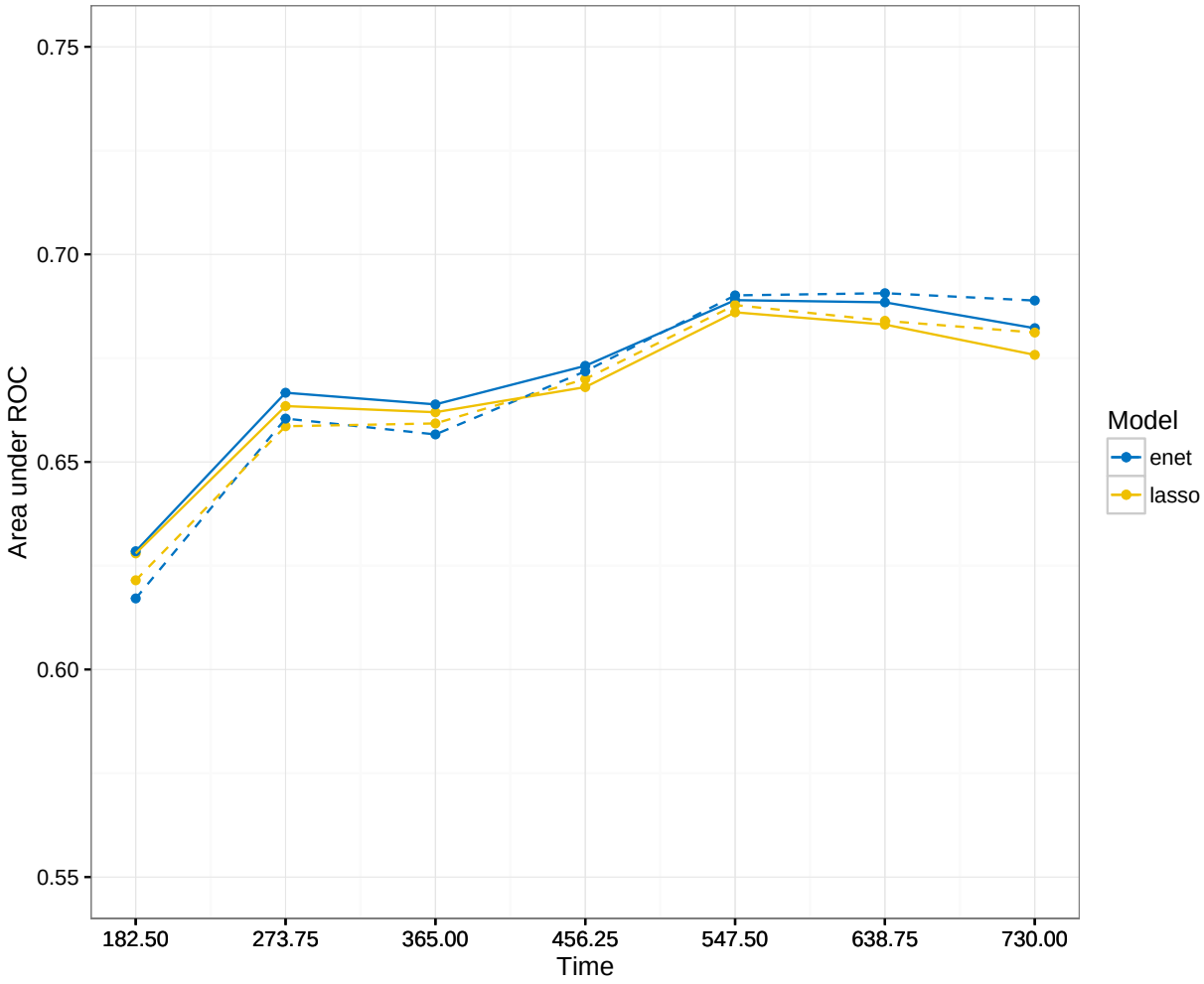

Figure 8: Model comparison results for lasso and elastic-net by validation

```
                        seed = 42, trace = FALSE)
```

```
print(cmp.cal)
```

```
## High-Dimensional Cox Model Calibration Object
## Random seed: 42
## Calibration method: repeated cross-validation
## Cross-validation folds: 10
## Cross-validation repeated times: 10
## Model type: lasso
## glmnet model alpha: 1
## glmnet model lambda: 0.01952497
## glmnet model penalty factor: not specified
## Calibration time point: 3285
## Number of groups formed for calibration: 5
##
## High-Dimensional Cox Model Calibration Object
## Random seed: 42
## Calibration method: repeated cross-validation
## Cross-validation folds: 10
## Cross-validation repeated times: 10
## Model type: enet
## glmnet model alpha: 0.1
## glmnet model lambda: 0.1117291
## glmnet model penalty factor: not specified
## Calibration time point: 3285
## Number of groups formed for calibration: 5
```

```
summary(cmp.cal)
```

```
##   Model type: lasso
##   Calibration Summary Table
##   Predicted  Observed Lower 95% Upper 95%
## 1 0.5963343 0.4754581 0.4022602 0.5619756
## 2 0.7004063 0.6891303 0.5691624 0.8343848
## 3 0.7516920 0.8162030 0.7698701 0.8653244
## 4 0.7888483 0.7929648 0.6943699 0.9055595
## 5 0.8283105 0.9006215 0.8684128 0.9340248
##
##   Model type: enet
##   Calibration Summary Table
##   Predicted  Observed Lower 95% Upper 95%
## 1 0.6120541 0.4887871 0.4163319 0.5738520
## 2 0.7070683 0.6854374 0.5795487 0.8106730
## 3 0.7522133 0.8251830 0.7787630 0.8743700
## 4 0.7848643 0.7877938 0.6880305 0.9020225
## 5 0.8181001 0.9109381 0.8788793 0.9441664
```

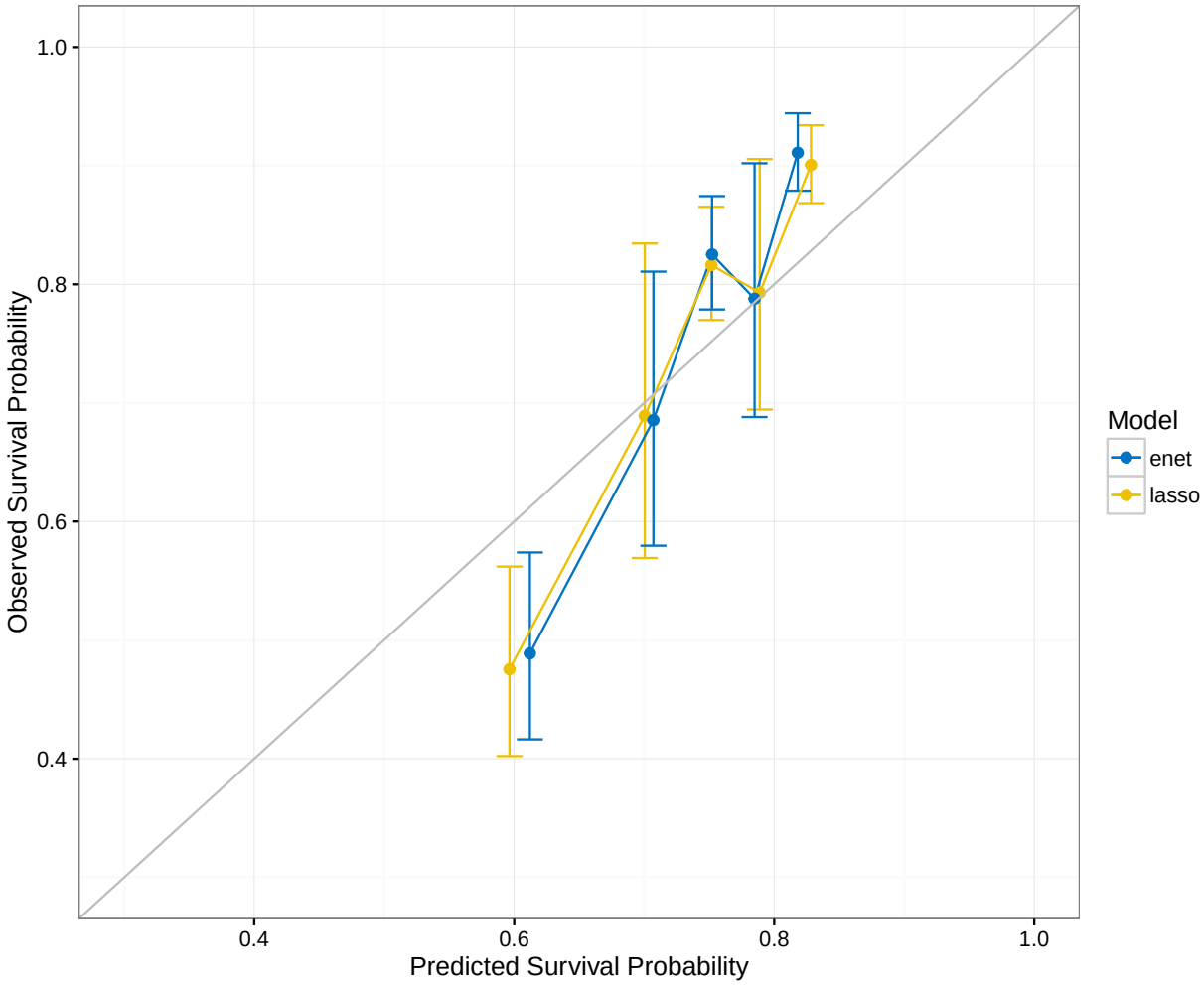

Figure 9: Model comparison results for lasso and elastic-net by calibration

```
plot(cmp.cal, xlim = c(0.3, 1), ylim = c(0.3, 1))
```

The summary output and the plot gave the calibration results for each model type we want to compare. Lasso and elastic-net have comparable performance in this case, since their predicted overall survival probabilities are both close to the observed survival probabilities in a similar degree.

7 Prediction on New Data

To predict the overall survival probability on certain time points for new samples with the established models, simply use `predict()` on the model objects and the new data.

As an example, we will use the samples numbered from 101 to 105 in the `smart` dataset as the new samples, and predict their overall survival probability from one year to ten years:

```
predict(aenetfit, x, y, newx = x[101:105, ], pred.at = 1:10 * 365)
```

```
##             365       730       1095      1460      1825      2190      2555
## [1,] 0.9476831 0.9203352 0.8883491 0.8572830 0.8171430 0.7841366 0.7430376
## [2,] 0.9714625 0.9562557 0.9382028 0.9203799 0.8969040 0.8771987 0.8521190
## [3,] 0.9786224 0.9671660 0.9535052 0.9399529 0.9220000 0.9068385 0.8874163
## [4,] 0.8971762 0.8456666 0.7873705 0.7327622 0.6651361 0.6120032 0.5489623
## [5,] 0.9736111 0.9595251 0.9427806 0.9262253 0.9043816 0.8860127 0.8625882
##            2920      3285      3650
## [1,] 0.7047801 0.6547589 0.6547589
## [2,] 0.8281917 0.7959836 0.7959836
## [3,] 0.8687509 0.8434080 0.8434080
## [4,] 0.4933861 0.4252340 0.4252340
## [5,] 0.8401909 0.8099641 0.8099641
```

8 Customize Color Palette

The `hdnom` package has 4 unique built-in color palettes available for all above plots, inspired by the colors commonly used by scientific journals. Users can use the `col.pal` argument to select the color palette. Possible values for this argument are listed in the table below:

Value	Color Palette Related Journals
`"JCO"`	*Journal of Clinical Oncology*
`"Lancet"`	Lancet journals, such as *Lancet Oncology*
`"NPG"`	NPG journals, such as *Nature Reviews Cancer*
`"AAAS"`	AAAS Journals, such as *Science*

By default, `hdnom` will use the JCO color palette (`col.pal = "JCO"`).

References

Harrell, Frank. 2015. *Regression Modeling Strategies: With Applications to Linear Models, Logistic and Ordinal Regression, and Survival Analysis*.

Steyerberg, Ewout W. 2008. *Clinical Prediction Models: A Practical Approach to Development, Validation, and Updating*. Springer Science & Business Media.