Model evaluation and analysis: the *modEvA* R package in a nutshell

A. Márcia Barbosa, CIBIO/InBIO - University of Évora (Portugal), barbosa@uevora.pt

updated 5 Mar 2015

The ***modEvA*** package works within the free and open-source R statistical software, so you first need to **download, install and open R** (available at http://www.r-project.org). In this tutorial, in `monospaced` font are the commands that you need to type (or copy and paste) into the R console (and then press the *enter* key to execute them). For commands that generate visible results in R, these are usually shown below them, preceded by hash marks (##). Note that all **commands are case-sensitive**, so you must respect upper- and lower-case letters; that you must always use **straight** (', ") **rather than curly quotes and apostrophes**; and that **R is only ready to receive a new command when there's a prompt sign (>) at the end of the R console**; if not, it's still waiting for an operation to be finished or for you to complete a previous command -- watch out for unclosed parentheses or such.

Install *modEvA* by pasting the command below in the R console (when connected to the internet):

```
install.packages("modEvA", repos = "http://R-Forge.R-project.org")
```

This should work if you have the **latest version of R**; otherwise, it may either fail (producing a message like "*package 'modEvA' is not available for your R version*") or show a warning and install an older version of *modEvA*. To **check the version that you have actually installed**, type `citation(package="modEvA")`. To install the latest version of the package, you can either upgrade R *or* download the compressed *modEvA* package **source files** to your disk -- *.zip* for Windows or *.tar.gz* for Linux and Mac, available at the package development page or at this Dropbox folder and then install the package locally, e.g. with R menu "*Packages - Install packages from local zip files*" (Windows), or "*Packages & Data - Package installer, Packages repository - Local source package*" (Mac), or "*Tools - Install packages - Install from: Package Archive File*" (RStudio).

You only need to install the package once (unless a new version becomes available), but you need to **load it every time you open a new R session** in which you intend to use *modEvA* (no need for an internet connection anymore), by pasting the following command in R:

```
library(modEvA)
```

Load the *rotif.mods* sample dataset that comes with the *modEvA* package, to use as an example:

```
data(rotif.mods)
```

You can get more information on this dataset (the following command should open an R Documentation window):

```
help(rotif.mods)
```

You can see that *rotif.mods* is a list containing two elements: a list of generalized linear *models* and a dataframe with their *predictions*. Let's leave the predictions alone for now, and work on the model objects. Let's start by checking out their names:

```
names(rotif.mods$models)
```

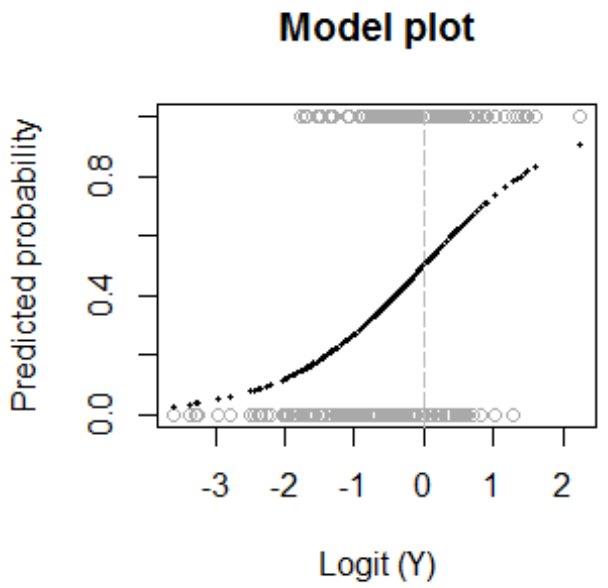
```
##  [1] "Abrigh" "Afissa" "Apriod" "Bangul" "Bcalyc" "Bplica" "Bquadr"
##  [8] "Burceo" "Cgibba" "Edilat" "Flongi" "Kcochl" "Kquadr" "Ktropi"
## [15] "Lbulla" "Lclost" "Lhamat" "Lluna"  "Llunar" "Lovali" "Lpatel"
## [22] "Lquadr" "Mventr" "Ppatul" "Pquadr" "Pvulga" "Specti" "Tpatin"
## [29] "Tsimil" "Ttetra"
```

These names correspond to species abbreviations, and each object in this list is a generalized linear model of the presence-absence of the corresponding species. Now let's assign the first model to an individual object, and use it to try out *modEvA* functions. The following two commands should produce the same result, as "Abrigh" is the name of the 1st model in the *rotif.mods$models* list:

```
mod <- rotif.mods$models[[1]]
mod <- rotif.mods$models[["Abrigh"]]
```

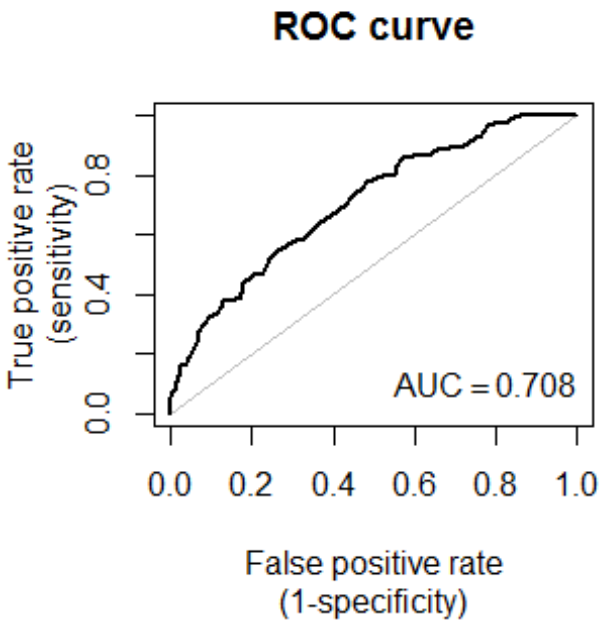
Let's now try the *modEvA* functions on this model (or you can use any other generalized linear model that you may have). Let's start with *plotGLM*, which shows how observed (grey) and predicted (black) values vary along the regression equation:

```
plotGLM(model = mod, xlab = "Logit (Y)", ylab = "Predicted probability", main =
"Model plot")
```



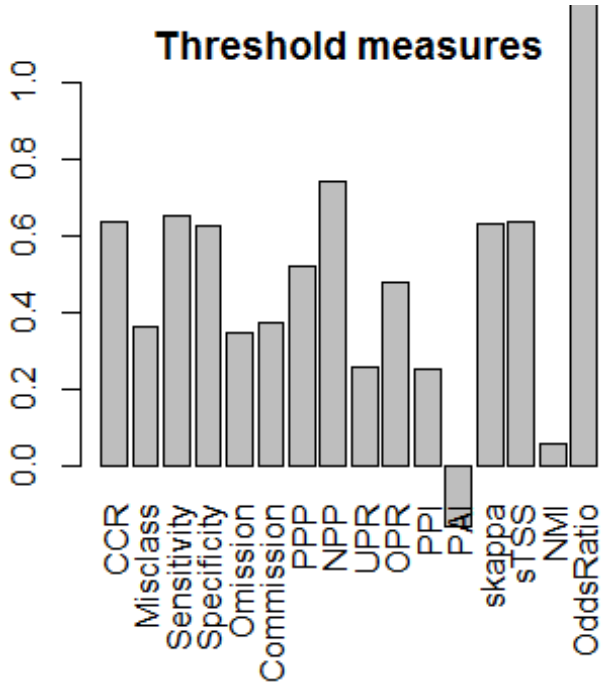
Now calculate the area under the ROC curve for this model. This function produces a plot and also some text results, which will appear in your R console but will not be shown here:

```
AUC(model = mod)
```



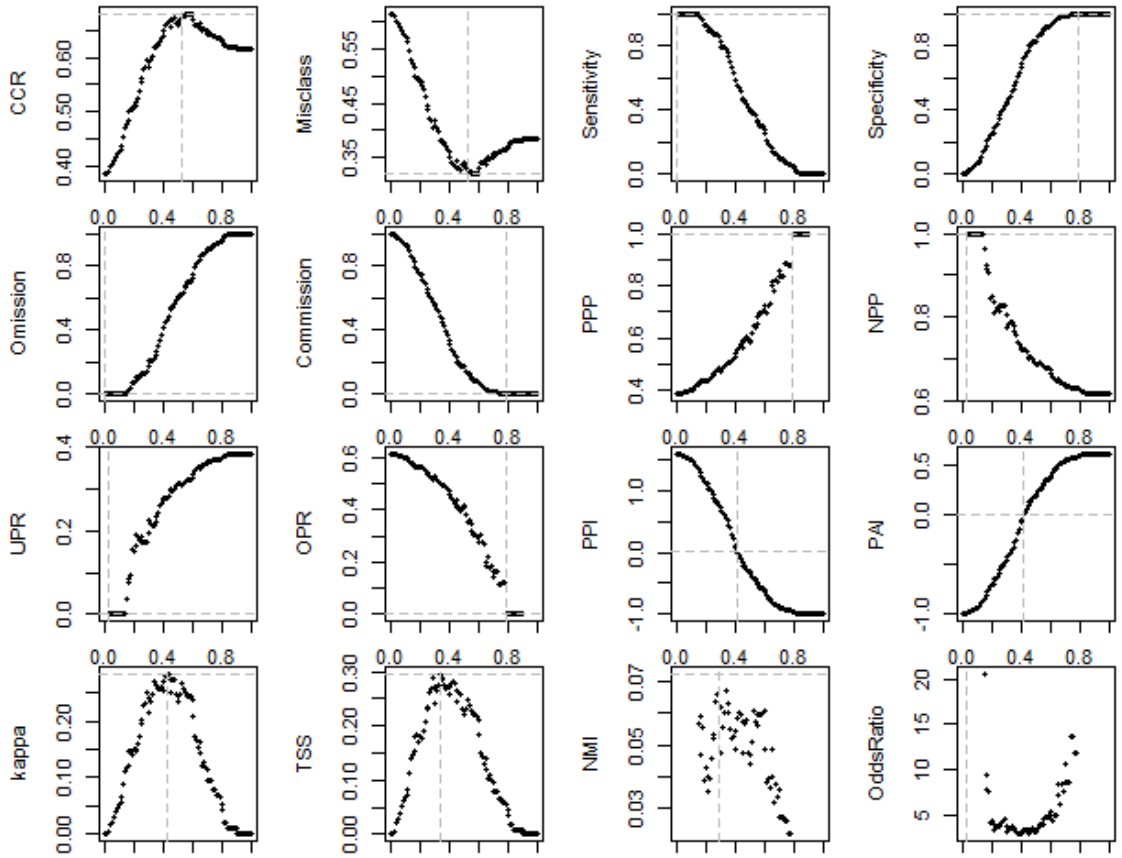
Calculate some threshold-based measures for this model, using the species' prevalence (proportion of presences) as the threshold value above which to consider that the model predicts the species to be present (again, only the plot is shown here, but text results should appear in your R console):

```
par(mar = c(5.6, 4.1, 2, 2.1))
threshMeasures(model = mod, thresh = "preval", ylim = c(0, 1), main =
"Threshold measures")
```



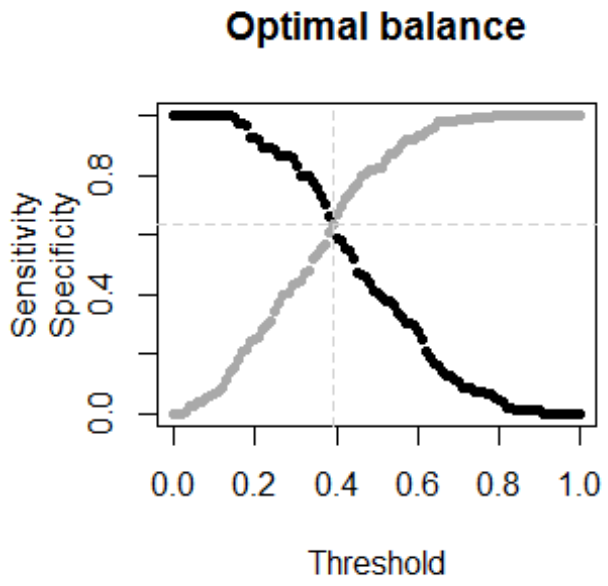
Now see how each threshold-based measure varies along with the chosen prediction threshold, to maybe identify optimal thresholds:

```
optiThresh(model = mod, pch = 20)
```



You can also calculate the optimal threshold balancing two complementary evaluation measures:

```
optiPair(model = mod, measures = c("Sensitivity", "Specificity"), main =
"Optimal balance")
```



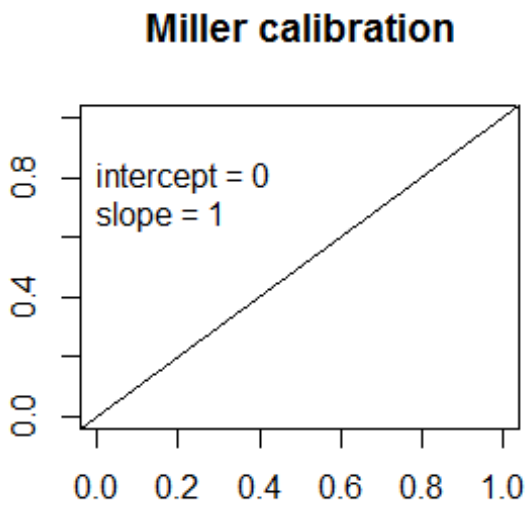
(You can try this with other pairs of related *measures*, such as `c("Omission", "Commission")`, `c("PPI", "PAI")`, etc.).

Now let's assess the proportion of variation that the model accounts for. For GLMs there isn't a single consensual measure for this; *modEvA* can calculate the explained deviance (*D-squared*), optionally adjusted for the number of observations and parameters; and some pseudo R-squared values (see `help(Dsquared)` and `help(RsqGLM)` for further info):

```
Dsquared(model = mod)

## [1] 0.1114199

Dsquared(model = mod, adj = TRUE)

## [1] 0.09264705

RsqGLM(model = mod)

## $CoxSnell
## [1] 0.1380008
##
## $Nagelkerke
## [1] 0.187434
##
## $McFadden
## [1] 0.1114199
##
## $Tjur
## [1] 0.1365661
##
## $sqPearson
## [1] 0.134168
```

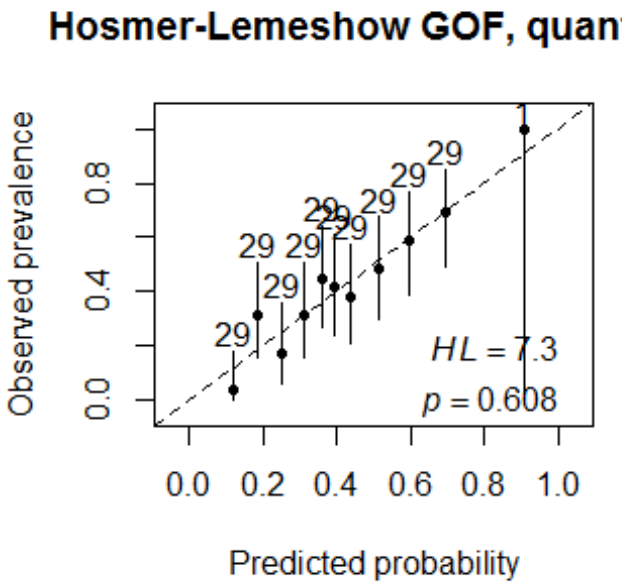
We can also take a look at some model calibration measures, such as Miller's calibration statistics and the Hosmer-Lemeshow goodness-of-fit. Note, however, that the former is not useful for evaluating a model on the same data used for building it (the results will always look good); and that the latter depends strongly on the *bin.method* used for grouping the values, as you can see below. See `help(MillerCalib)` and `help(HLfit)` to find out how these measures are calculated.

```
MillerCalib(model = mod)
```



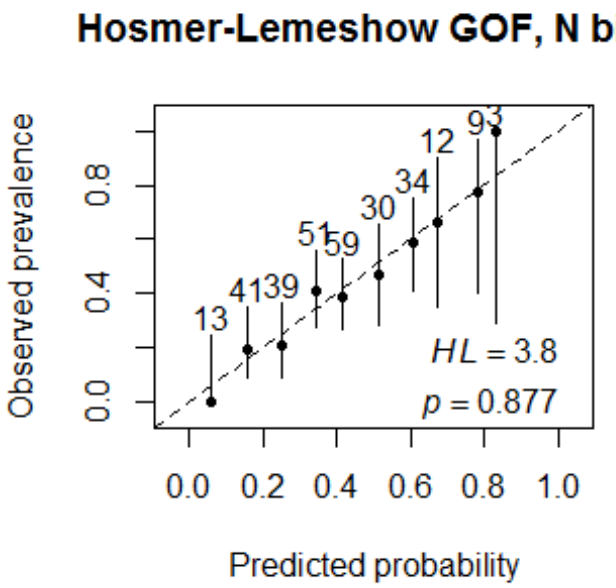

```
## $intercept
## [1] 1.799596e-13
##
## $slope
## [1] 1
```

```
HLfit(model = mod, bin.method = "quantiles", main = "Hosmer-Lemeshow GOF,
quantiles")
```




```
## $bins.table
##    BinCenter NBin     BinObs   BinPred BinObsCIlower BinObsCIupper
## 1  0.1194811   29 0.03448276 0.1016246  0.0008726469     0.1776443
## 2  0.1819756   29 0.31034483 0.1789304  0.1528459396     0.5083234
## 3  0.2482370   29 0.17241379 0.2445406  0.0584560830     0.3577476
## 4  0.3084812   29 0.31034483 0.3091572  0.1528459396     0.5083234
## 5  0.3618102   29 0.44827586 0.3607436  0.2644553037     0.6430613
## 6  0.3927488   29 0.41379310 0.3942488  0.2352402098     0.6106372
## 7  0.4383741   29 0.37931034 0.4372018  0.2068686995     0.5773954
## 8  0.5130304   29 0.48275862 0.5063462  0.2944855830     0.6746850
## 9  0.5960986   29 0.58620690 0.5910230  0.3893627914     0.7647598
## 10 0.6933976   29 0.68965517 0.7070658  0.4916766462     0.8471541
## 11 0.9044237    1 1.00000000 0.9044237  0.0250000000     1.0000000
##
## $chi.sq
## [1] 7.278077
##
## $DF
## [1] 9
##
## $p.value
## [1] 0.6081922
```

```
HLfit(model = mod, bin.method = "n.bins", main = "Hosmer-Lemeshow GOF, N bins")
```




```
## $bins.table
##                BinCenter NBin    BinObs    BinPred BinObsClower
## (0.026,0.115] 0.05859488   13 0.0000000 0.06339589    0.00000000
## (0.115,0.202] 0.15753130   41 0.1951220 0.15789124    0.08820610
## (0.202,0.29]  0.24856387   39 0.2051282 0.24747539    0.09296393
## (0.29,0.378]  0.34035838   51 0.4117647 0.33974064    0.27584296
## (0.378,0.466] 0.41206526   59 0.3898305 0.41510104    0.26549147
## (0.466,0.553] 0.51322862   30 0.4666667 0.50772771    0.28341808
## (0.553,0.641] 0.60663508   34 0.5882353 0.60041852    0.40696943
## (0.641,0.729] 0.66899210   12 0.6666667 0.67465548    0.34887551
## (0.729,0.817] 0.78101033    9 0.7777778 0.77077823    0.39990643
## (0.817,0.905] 0.83199370    3 1.0000000 0.85136910    0.29240177
##               BinObsCIupper
## (0.026,0.115]     0.2470526
## (0.115,0.202]     0.3486655
## (0.202,0.29]      0.3646442
## (0.29,0.378]      0.5583072
## (0.378,0.466]     0.5256215
## (0.466,0.553]     0.6567448
## (0.553,0.641]     0.7535293
## (0.641,0.729]     0.9007539
## (0.729,0.817]     0.9718550
## (0.817,0.905]     1.0000000
##
## $chi.sq
## [1] 3.770615
##
## $DF
## [1] 8
##
## $p.value
## [1] 0.8772036
```

You can calulate a set of evaluation measures for several models simultaneously, if you have them in a list like *rotif.mods$models* (results not shown here):

```
multModEv(models = rotif.mods$models, thresh = "preval", bin.method =
"quantiles")
```

Most of the *modEvA* functions also have *obs* and *pred* arguments, so instead of *model* objects you can use vectors of observed and predicted values as input. You can find out additional options and further info on any function with *help(function.name)*.

Some other potentially useful *modEvA* functions are (still) not covered in this tutorial, but you can find out about them as well:

```
help(evenness)
help(prevalence)
help(OA)
help(MESS)
help(varPart)
```

That's it! E-mail me with any suggestions or concerns, but first remember to check for updates to the package or this tutorial at http://modEvA.r-forge.r-project.org. This tutorial was built with *RStudio* + *rmarkdown* + *knitr*. Thanks!