

Daniel Nee



MACHINE LEARNING

CALIBRATING CLASSIFIER PROBABILITIES

OCTOBER 9, 2014 | DN | 6 COMMENTS

You've built your classifier, run cross-validation and have a super high AUC. So you are done right? Maybe not.

Most classifiers output a score of how likely an observation is to be in the positive class. Usually these scores are between 0 and 1 and get called probabilities. However these probabilities often do not reflect reality, e.g. a probability of 20% may not mean it has a 20% chance of happening, it could have a 10%, 50%, 70%, etc. chance of happening. Our aim should be that our model outputs accurately reflect posterior probabilities $P(Y = 1|x)$.

In the post we will mainly focus on binary classifiers. Later in the post will will talk about how to extend these ideas to mutliclass problems.

Why it happens

Our models can output inaccurate probabilities for a variety of reasons:

- Flawed model assumptions (e.g. independence in a Naive Bayes Model)
- Hidden features not available at training time
- Deficiencies in the learning algorithm.

In terms of learning algorithms, as noted in [Niculescu-Mizil et al](#) and through my own research:

- Naive Bayes tends to push probabilities toward the extremes of 0 and 1.
- SVMs and boosted trees tend to push probabilities away from 0 and 1 (toward the centre)
- Neural networks and random forests tend to have well calibrated probabilities.
- Regularisation also tends to push probabilities toward the centre.

Do we care?

Whether or not we want well calibrated probabilities depends entirely on the problem we are trying to solve.

If we only need to rank observations from most likely to least likely, then calibration is unnecessary.

Examples of problems I have worked on where calibrated probabilities are extremely important:

- Loan default prediction - Banks will generally be setting thresholds on the probabilities, auto-reject if probability of default is above 30%, etc.
- Ad Click Prediction - Decided what ad to show, how much to bid. You might use a baseline Click Through Rate (CTR), and compare your prediction to this to see how much more you are willing to pay for this ad impression. ¹
- Demographics Estimation of Websites - Imagine you have predictions of gender/ages, as probabilities, for a number of web users. Estimating the gender distribution on a website, can be done by just averaging the probabilities of the users seen on the site. Any bias in the probabilities, will generate a bias in your estimate.

Visualisation

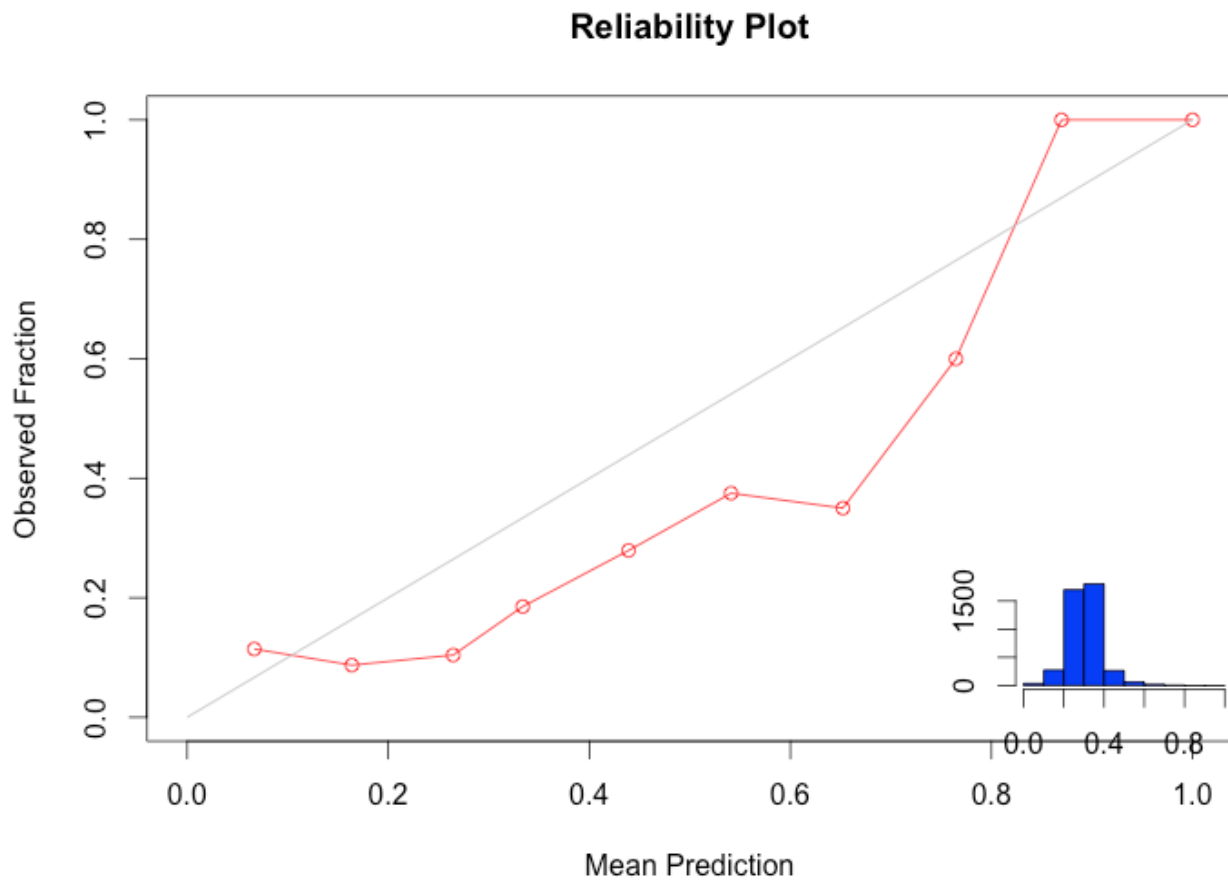
A **reliability diagram** is a relatively simple technique for visualising how well calibrated our classifier is. As described in [Niculescu-Mizil et al](#):

On real problems where the true conditional probabilities are not known, model calibration can be visualized with reliability diagrams (DeGroot & Fienberg, 1982). First, the prediction space is discretized into ten bins. Cases with predicted value between 0 and 0.1 fall in the first bin, between 0.1 and 0.2 in the second bin, etc. For each bin, the mean predicted value is plotted against the true fraction of positive cases. If the model is well calibrated the points will fall near the diagonal line.

Below we provide a piece of R code for producing reliability diagrams. Here we generalise the number of bins to be a user defined parameter.

```
1 reliability.plot <- function(obs, pred, bins=10, scale=T) {  
2   # Plots a reliability chart and histogram of a set of predictions from a classifier  
3   #  
4   # Args:  
5   #   obs: Vector of true labels. Should be binary (0 or 1)  
6   #   pred: Vector of predictions of each observation from the classifier. Should be a  
7   #         number  
8   #   bins: The number of bins to use in the reliability plot  
9   #   scale: Scale the pred to be between 0 and 1 before creating reliability plot  
10  require(plyr)  
11  library(Hmisc)  
12  
13  min.pred <- min(pred)  
14  max.pred <- max(pred)  
15  min.max.diff <- max.pred - min.pred  
16  
17  if (scale) {  
18    pred <- (pred - min.pred) / min.max.diff  
19  }  
20  
21  bin.pred <- cut(pred, bins)  
22  
23  k <- ldply(levels(bin.pred), function(x) {  
24    idx <- x == bin.pred  
25    c(sum(obs[idx]) / length(obs[idx]), mean(pred[idx]))  
26  })  
27  
28  is.nan.idx <- !is.nan(k$V2)  
29  k <- k[is.nan.idx,]  
30  plot(k$V2, k$V1, xlim=c(0,1), ylim=c(0,1), xlab="Mean Prediction", ylab="Observed Fr  
31  lines(c(0,1),c(0,1), col="grey")  
32  subplot(hist(pred, xlab="", ylab="", main="", xlim=c(0,1), col="blue"), grconvertX(c  
33 }
```

In the figure below we show an example reliability plot. Ideally the reliability plot (red line) should be as close to the diagonal line as possible. As there is significant deviation from the diagonal, calibrating the probabilities will possibly help.



It is also worth mentioning that if you take the mean of the score distribution, it should ideally be close to the prior.

Techniques for calibration

Overfitting

The most important step is to create a separate dataset to perform calibration with. Our steps for calibration are:

- Split dataset into test and train
- Split the train set into model training and calibration.
- Train the model on train set
- Score test and calibration set
- Train the calibration model on calibration set
- Score the test set using calibration

How much data to use for calibration will depend on the amount of data you have available. The calibration model will generally only be fitting a small number of parameters (so you do not need a huge volume of data). I would aim for around 10% of your training data, but at a minimum of at least 50 examples.

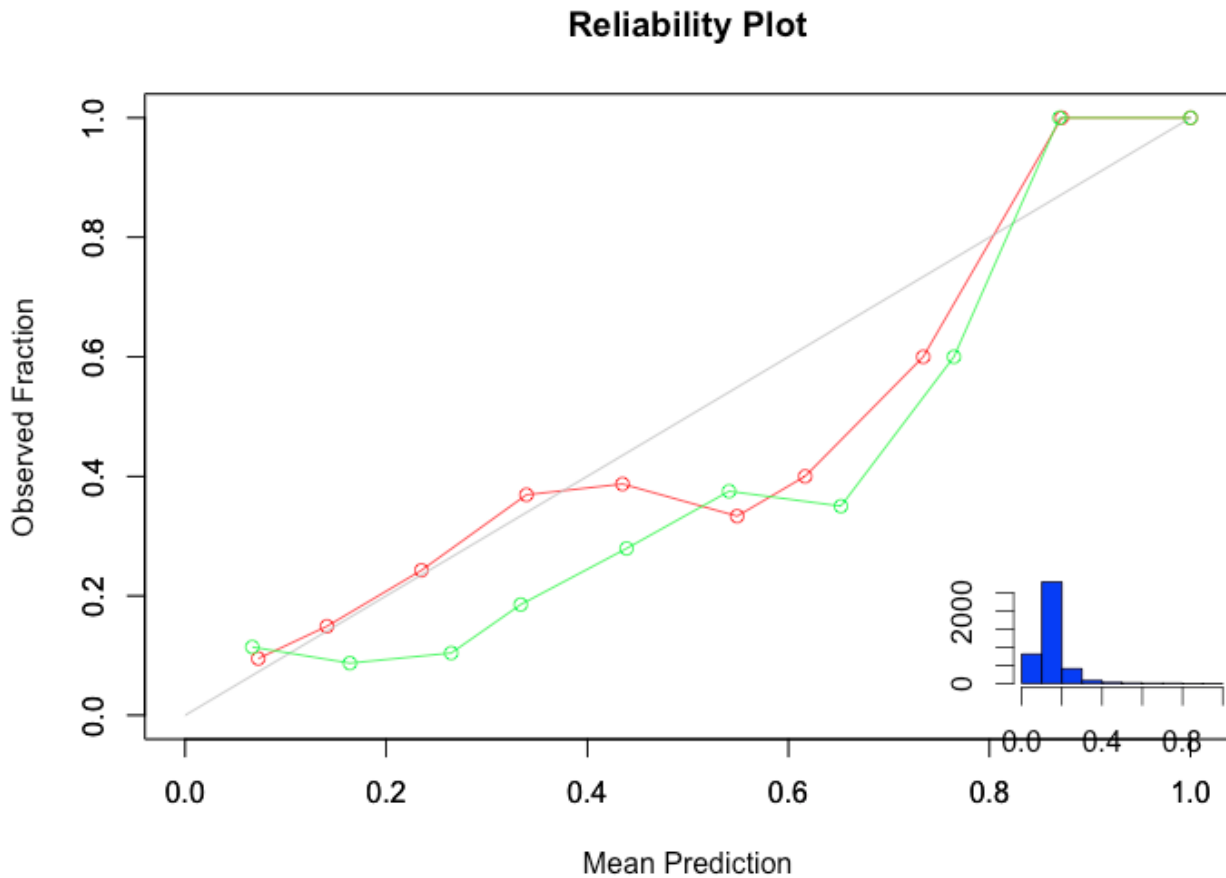
Platt Scaling

Platt Scaling essentially involves fitting a logistic regression on the classifier output. Originally developed to fit probabilities to the outputs of SVM², it is also well suited to the output of most other classifiers.

```
1 calib.data.frame <- data.frame(cbind(Y.calib, Y.calib.pred))
2 colnames(calib.data.frame) <- c("y", "x")
3 calib.model <- glm(y ~ x, calib.data.frame, family=binomial)
4 calib.data.frame <- data.frame(Y.test.pred)
5 colnames(calib.data.frame) <- c("x")
6 Y.test.pred.calibrated <- predict(calib.model, newdata=calib.data.frame, type="response")
```

The reliability diagram below shows the original reliability plot (green) and after Platt Scaling (red).

The Platt Scaling should not change the rank of the observations, so measures such as AUC will be unaffected. However, measures like Log Loss³ will be improved. In this example, Log Loss was originally 0.422 and improved to 0.418.



In Platt's original paper suggests , instead of using the original $\{0,1\}$ targets in the calibration sample, it suggests to mapping to:

$$t_+ = \frac{N_+ + 1}{N_+ + 2}$$

$$t_- = \frac{1}{N_- + 2}$$

where N_+ and N_- are the number of positive and negative examples in the calibration sample.

To some extent this introduces a level of regularisation. Imagine if you only gave probabilities of either 0 or 1 and you correctly predicted all examples. Your Log Loss would be zero. With Platt's transformation, you Log Loss would be non-zero. As the Log Loss is what you are optimising when fitting the logistic regression, a level of regularisation is introduced.

In my experiments, this transformation had little to no effect on the reliability diagram and Log Loss, so seems an unnecessary step. It may be useful if you have very few examples and overfitting is more of a concern (therefore regularisation would help). You could also use a ridge or lasso regression.

Isotonic Regression

With Isotonic Regression you make the assumption:

$$y_i = m(f_i) + \epsilon_i$$

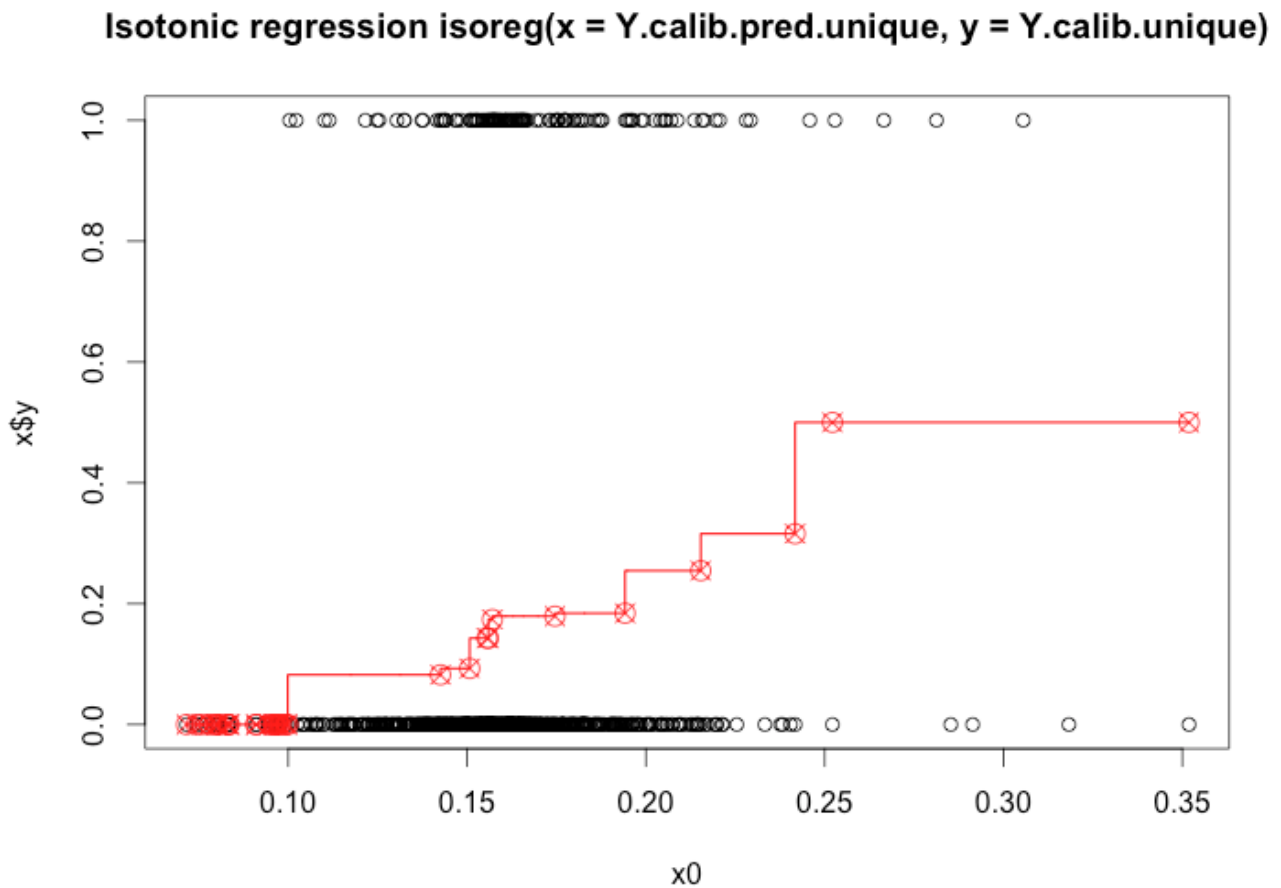
where m is an isotonic (monotonically increasing or decreasing) function. This is the exact same assumptions we would use for least squares, except m is now a isotonic function instead of linear.

Below is an R example of how to perform isotonic regression using the `isoreg` function.

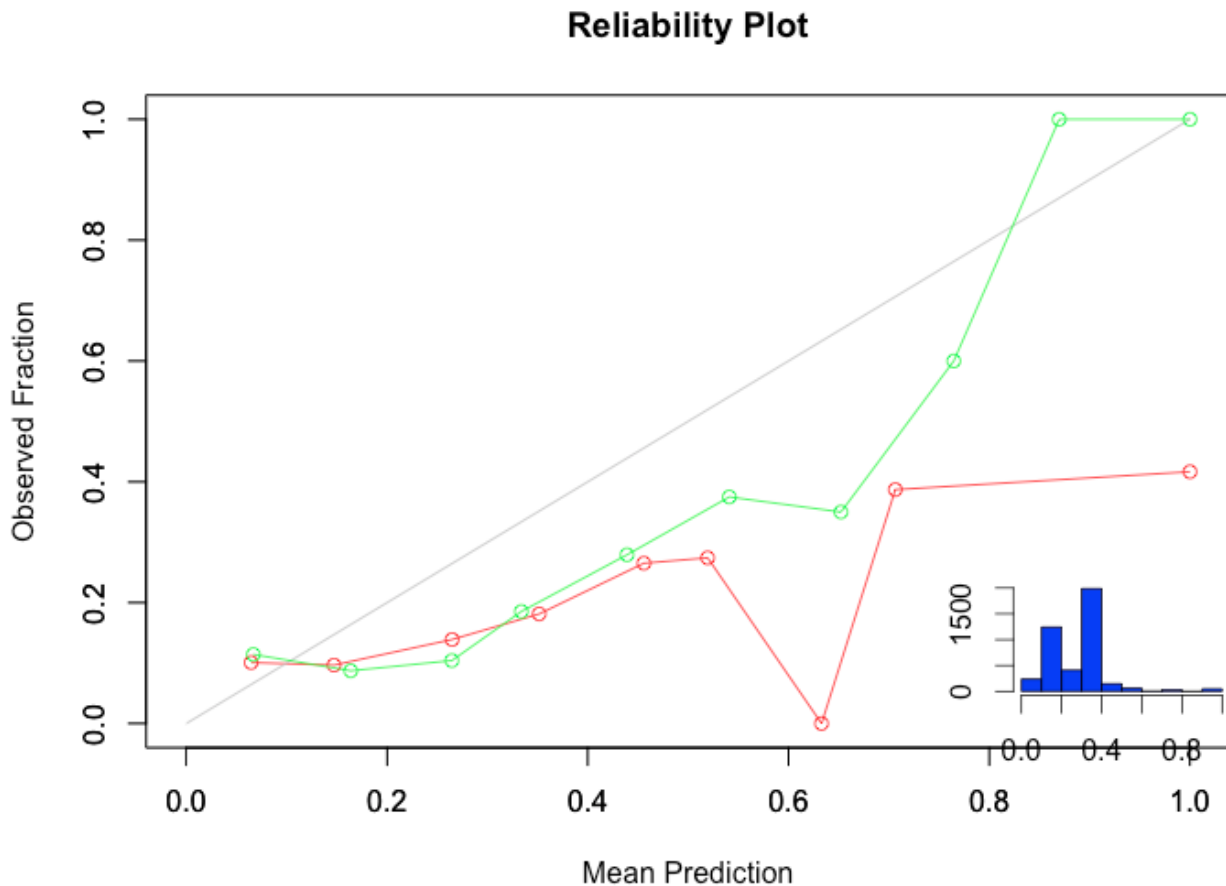
```
1 fit.isoreg <- function(iso, x0)
2 {
3   o = iso$o
4   if (is.null(o))
5     o = 1:length(x)
6   x = iso$x[o]
7   y = iso$yf
8   ind = cut(x0, breaks = x, labels = FALSE, include.lowest = TRUE)
9   min.x <- min(x)
10  max.x <- max(x)
11  adjusted.knots <- iso$iKnots[c(1, which(iso$yf[iso$iKnots] > 0))]
12  fits = sapply(seq(along = x0), function(i) {
13    j = ind[i]
14
15    # Handles the case where unseen data is outside range of the training data
16    if (is.na(j)) {
17      if (x0[i] > max.x) j <- length(x)
18      else if (x0[i] < min.x) j <- 1
19    }
20
21    # Find the upper and lower parts of the step
22    upper.step.n <- min(which(adjusted.knots > j))
23    upper.step <- adjusted.knots[upper.step.n]
24    lower.step <- ifelse(upper.step.n==1, 1, adjusted.knots[upper.step.n -1] )
25
26    # Perform a linear interpolation between the start and end of the step
27    denom <- x[upper.step] - x[lower.step]
28    denom <- ifelse(denom == 0, 1, denom)
29    val <- y[lower.step] + (y[upper.step] - y[lower.step]) * (x0[i] - x[lower.step]) /
30
31    # Ensure we bound the probabilities to [0, 1]
32    val <- ifelse(val > 1, max.x, val)
33    val <- ifelse(val < 0, min.x, val)
34    val <- ifelse(is.na(val), max.x, val) # Bit of a hack, NA when at right extreme of
35    val
36  })
37  fits
38 }
39
40 # Remove any duplicates
41 idx <- duplicated(Y.calib.pred)
42 Y.calib.pred.unique <- Y.calib.pred[!idx]
43 Y.calib.unique <- Y.calib[!idx]
44
45 iso.model <- isoreg(Y.calib.pred.unique, Y.calib.unique)
46
```

```
47 Y.test.pred.calibrated <- fit.isoreg(iso.model, Y.test.pred)
```

In the figure below we show an example of the sort of function fitted by the isotonic regression model:



Notice how it goes up in steps instead of a smooth curve. To smooth the fit, we perform a linear interpolation between each step.



In the reliability plot above, the original uncalibrated scores are shown in green and the isotonic regression scores are shown in red. In this example we find isotonic regression actually made it worse. The Log Loss for instance went from 0.422 to 0.426. The AUC was also reduced.

Multiclass Classification




What happens if you have more than two classes? Firstly I would recommend visualising the problem as a series of reliability diagrams. For k classes, you can create k reliability diagrams.

Secondly, you can take the score for each of your k classes and plug them into a multinomial logistic regression. The superb **glmnet** package implements a multinomial logistic regression. You can set the regularisation parameter to something quite small. One word of caution, if you have many classes, overfitting can become an issue. At this point it is worth optimising the regularisation parameter.

If your favourite machine learning model (e.g. SVM) doesn't directly support multi-class classification, you can fit a 1 vs. all set of classifiers and then plug each of those scores into the multinomial logistic regression.

Summary

Classifier probability calibration can be an important step in your machine learning pipeline. The first step is to always visualise and see how much of an issue you have. In general I have found Platt Scaling to be the simplest and most effective approach to most calibration of classification problems.

1. See Google's paper on issues with ad click prediction: <http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/41159.pdf> 
2. See Platt's original paper: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.41.1639> 
3. Log loss definition: <https://www.kaggle.com/wiki/LogarithmicLoss> 

 CALIBRATION  CLASSIFIER  ISOTONIC REGRESSION  PLATT SCALING  RELIABILITY DIAGRAM

6 THOUGHTS ON “CALIBRATING CLASSIFIER PROBABILITIES”



Pablo

OCTOBER 22, 2015 AT 2:25 PM

Hi Daniel,

IS IT POSSIBLE TO DO A PARAMETER EXPLORATION WHERE THE SCORE FUNCTION (MAKE_SCORE) IS SOMETHING THAT CALIBRATES THE PROBABILITIES NOT (NECESSARILY) THE BINARY CLASSIFICATION??

SO IT WILL SEARCH FOR A COMBINATIONS OF PARAMETERS FOR A CERTAIN MODEL TRAIN-CALIBRATE AND THEN CHECK IF THE CALIBRATION IS CORRECT IN THE TEST SET.

I think that the question is better explained in this post:

http://stats.stackexchange.com/questions/178125/machine-learning-how-do-we-usually-select-the-best-combination-of-parameters-to?noredirect=1#comment337351_178125



★ **dn**

NOVEMBER 12, 2015 AT 1:59 PM

Hi Pablo,

If I have understood correctly, I think the answer is yes.

How I would set it up is, split the data into train and calibration. Build your model on the train. Consider that model now fixed and don't change it (this will mean your calibration step will never change the AUC performance of your model).

By then minimising the Log Loss on the calibration set (apply CV on the calibration data), you will be improving the how well calibrated it is. So in your reliability plots, the lines should get closer to the diagonal.

Dan



Diego de Las Casas

JANUARY 19, 2016 AT 11:21 PM

Hi,

Thanks for the post! I can't find references to isotonic regression for calibration using R anywhere...

I used your code as a base for mine, but found a bug in the `fit.isoreg` function and thought you should know about it:

In line 22 you wrote:

```
upper.step.n <- max(which(abs(adjusted.knots-j)==min(abs(adjusted.knots-j))))+1
```

This does not work, sometimes the lower step will be closer than the upper step and the value will be assigned to a step lower.

I it changed to:

```
upper.step.n j))
```

which always get the smallest adjusted.knot that is bigger than j.

Hope it helps!

Thanks again,
Diego



Diego de Las Casas

JANUARY 19, 2016 AT 11:24 PM

Hi,

Thanks for the post! I can't find references to isotonic regression for calibration using R anywhere...

I used your code as a base for mine, but found a bug in the `fit.isoreg` function and thought you should know about it:

In line 22 you wrote:

```
upper.step.n <- max(which(abs(adjusted.knots-j)==min(abs(adjusted.knots-j))))+1
```

This does not work, sometimes the lower step will be closer than the upper step and the value will be assigned to a step lower.

I it changed to:

```
upper.step.n j))
```

which always get the smallest `adjusted.knot` that is bigger than `j`.

Hope it helps!

Thanks again,
Diego

P.S. -- for some reason my first comment did not have the right correction. Sorry for the recomment.



Diego de Las Casas

JANUARY 19, 2016 AT 11:29 PM

Got what is going wrong, the comment is being treated as html.

The line is:

```
upper.step.n = min(which(adjusted.knots > j))
```



★ dn

FEBRUARY 21, 2016 AT 3:17 PM

Thanks for spotting the bug Diego! I've updated the code