First, let’s start by installing and loading icarus and nnet, the two packages needed in this tutorial, from CRAN (if necessary):

install.packages(c("icarus","nnet"))

library(icarus)

library(nnet)

Then load the data:

load("data/weighting\_ML\_part1.RData")

The RData file contains two dataframes, one for the training set and one for the test set. They contain results of some international soccer games, from 01/2008 to 12/2016 for the training set, and from 01/2017 to 11/2017 for the test. Along with the team names and goals scored for each side, a few descriptive variables that we’re going to use as features of our ML models:

> head(train\_soccer)

Date team opponent\_team home\_field elo\_team

1 2010-10-12 Belarus Albania 1 554

2 2010-10-08 Bosnia and Herzegovina Albania 0 544

3 2011-06-07 Bosnia and Herzegovina Albania 0 594

4 2011-06-20 Argentina Albania 1 1267

5 2011-08-10 Montenegro Albania 0 915

6 2011-09-02 France Albania 0 918

opponent\_elo importance goals\_for goals\_against outcome year

1 502 1 2 0 WIN 2010

2 502 1 1 1 DRAW 2010

3 564 1 2 0 WIN 2011

4 564 1 4 0 WIN 2011

5 524 1 2 3 LOSS 2011

6 546 1 2 1 WIN 2011

*elo\_team* and *opponent\_elo* are quantitative variables indicative of the level of the team at the date of the game ; *importance* is a measure of high-profile the game played was (a friendly match rates 1 while a World Cup game rates 4). The other variables are imo self-descriptive.

Then we can train a multinomial logistic regression, with *outcome* being the predicted variable, and compute the predictions from the model:

outcome\_model\_unw <- multinom(outcome ~ elo\_team + opponent\_elo + home\_field + importance,

data = train\_soccer)

test\_soccer$pred\_outcome\_unw <- predict(outcome\_model\_unw, newdata = test\_soccer)

The sheer accuracy of this predictor is kinda good:

> ## Accuracy

> sum(test\_soccer$pred\_outcome\_unw == test\_soccer$outcome) / nrow(test\_soccer)

[1] 0.5526316

but it has a problem: it never predicts draws!

> summary(test\_soccer$pred\_outcome\_unw)

DRAW LOSS WIN

0 208 210

And indeed, draws being less common than other results, it seems more profitable for the algorithm that optimizes accuracy never to predict them. As a consequence, the probabilities of the game being a draw is always lesser than the probability of one team winning it. We could show that the probabilities are [not well calibrated](http://scikit-learn.org/stable/modules/calibration.html).

A common solution to this problem is to use reweighting to correct the imbalances in the sample, which we’ll now tackle. It is important to note that the weighting trick has to happen in the training set to avoid “data leaks”. A very good piece on this subject has been written by Max Kuhn [in the documentation of caret](https://topepo.github.io/caret/subsampling-for-class-imbalances.html).

[](https://topepo.github.io/caret/subsampling-for-class-imbalances.html)https://topepo.github.io/caret/

Commonly, you would do:

train\_soccer$weight <- 1

train\_soccer[train\_soccer$outcome == "DRAW",]$weight <- (nrow(train\_soccer)/table(train\_soccer$outcome)[1]) \* 1/3

train\_soccer[train\_soccer$outcome == "LOSS",]$weight <- (nrow(train\_soccer)/table(train\_soccer$outcome)[2]) \* 1/3

train\_soccer[train\_soccer$outcome == "WIN",]$weight <- (nrow(train\_soccer)/table(train\_soccer$outcome)[3]) \* 1/3

> table(train\_soccer$weight)

0.916067146282974 1.22435897435897

3336 1248

The draws are reweighted with a factor greater than 1 and the other games with a factor lesser than 1. This balances the predicted outcomes and thus improves the quality of the probabilities …

outcome\_model <- multinom(outcome ~ elo\_team + opponent\_elo + home\_field + importance,

data = train\_soccer,

weights = train\_soccer$weight)

test\_soccer$pred\_outcome <- predict(outcome\_model, newdata = test\_soccer)

> summary(test\_soccer$pred\_outcome)

DRAW LOSS WIN

96 167 155

… though at a loss in accuracy:

> ## Accuracy

> sum(test\_soccer$pred\_outcome == test\_soccer$outcome) / nrow(test\_soccer)

[1] 0.5263158

Now let’s look at the balance of our training sample on other variables:

> round(table(test\_soccer$importance) / nrow(test\_soccer),2)

1 2 3 4

0.26 0.08 0.54 0.12

> round(table(train\_soccer$importance) / nrow(train\_soccer),2)

1 2 3 4

0.56 0.08 0.23 0.12

It seems that the test set features a lot more important matches than the training set. Let’s look further, in particular at the dates the matches of the training set were played:

> round(table(train\_soccer$year) / nrow(train\_soccer),2)

2008 2009 2010 2011 2012 2013 2014 2015 2016

0.10 0.11 0.11 0.10 0.11 0.13 0.11 0.11 0.12

Thus the matches of each year between 2008 and 2016 have the same influence on the final predictor. A better idea would be to give the most recent games a slightly higher influence, for example by increasing their weight and thus reducing the weights of the older games:

nyears <- length(unique(train\_soccer$year))

year\_tweak <- rep(1/nyears,nyears) \* 1:nyears

year\_tweak <- year\_tweak \* 1/sum(year\_tweak) ## Normalization

> year\_tweak

[1] 0.02222222 0.04444444 0.06666667 0.08888889 0.11111111 0.13333333

[7] 0.15555556 0.17777778 0.20000000

We determine it is thus a good idea to balance on these two additional variables (*year* and *importance*). Now how should we do this? A solution could be to create an indicator variable containing all the values of the cross product between the variables *outcome*, *year* and *importance*, and use the same reweighting technique as before. But this would not be very practical and more importantly, some of the sub-categories would be nearly empty, making the procedure not very robust. A better solution is to use survey sampling calibration and Icarus 

train\_soccer$weight\_cal <- 1

importance\_pct\_test <- unname(

table(test\_soccer$importance) / nrow(test\_soccer),

)

marginMatrix <- matrix(, nrow = 0, ncol = 1) %>% ## Will be replaced by newMarginMatrix() in icarus 0.3.2

addMargin("outcome", c(0.333,0.333,0.333)) %>%

addMargin("importance", importance\_pct\_test) %>%

addMargin("year", year\_tweak)

train\_soccer$weight\_cal <- calibration(data=train\_soccer, marginMatrix=marginMatrix,

colWeights="weight\_cal", pct=TRUE, description=TRUE,

popTotal = nrow(train\_soccer), method="raking")

outcome\_model\_cal <- multinom(outcome ~ elo\_team + opponent\_elo + home\_field + importance,

data = train\_soccer,

weights = train\_soccer$weight\_cal)

test\_soccer$pred\_outcome\_cal <- predict(outcome\_model\_cal, newdata = test\_soccer)

icarus gives a summary of the calibration procedure in the log (too long to reproduce here). We then observe a slight improvement in accuracy compared to the previous reweighting technique:

> sum(test\_soccer$pred\_outcome\_cal == test\_soccer$outcome) / nrow(test\_soccer)

[1] 0.5478469

But more importantly we have reason to believe that the we improved the quality of the probabilities assigned to each event (we could check this using metrics such as the Brier score or calibration plots) 

It is also worth noting that some algorithms (especially those who rely on bagging, boosting, or more generally on ensemble methods) naturally do a good job at balancing samples. You could for example rerun the whole code and replace the logit regressions by boosted algorithms. You would then observe fewer differences between the unweighted algorithm and its weighted counterparts.

Stay tuned for the part 2, where we’ll show a trick to craft better probabilities (particularly for simulations) using external knowledge on probabilities.