

Last friday, Euro 2020, one of the biggest events in International soccer, was kicked off by the inaugural match between Italy and Turkey (Italy won it 3-0). Euros (short for European Championships) are usually held every 4 years, but because of he-who-must-not-be-named, last year's edition was postponed to this summer, while keeping the name "Euro 2020" (much like the [Tokyo Olympics](#)). 4-5 years ago, for Euro 2016, I basically wanted to try some cool methods based on [splines](#) on a real-world example. The model ended up [doing fairly well](#), and most importantly it was a lot of fun, so we decided to do it again this year! The model, from training to the web app, is entirely built in R.

Group Phase
Knockout Phase

Team	Round of 16	Quarter-finals	Semi-finals	Final	Winner
Belgium	97%	70%	50%	34%	22%
England	96%	59%	38%	26%	15%
France	86%	55%	37%	23%	14%
Portugal	79%	41%	23%	12%	6%
Netherlands	91%	50%	25%	12%	5%
Spain	80%	46%	21%	11%	5%
Croatia	76%	38%	20%	9%	4%
Switzerland	72%	36%	18%	8%	4%
Denmark	59%	33%	17%	8%	3%
Sweden	80%	39%	19%	8%	3%
Italy	91%	42%	18%	7%	3%

Click on this image to access the predictions!

How it works

The model is built on about 12.500 international football matches played since 2006. The main part actually consists in a combination of two models, one that predicts the number of goals scored by the reference team in a given game, the other one that predicts the number of goals they will allow. This basically models offense and defense for each team. A number of models for soccer have used similar ideas in the past, including FiveThirtyEight's brilliant [Soccer Power Index](#).

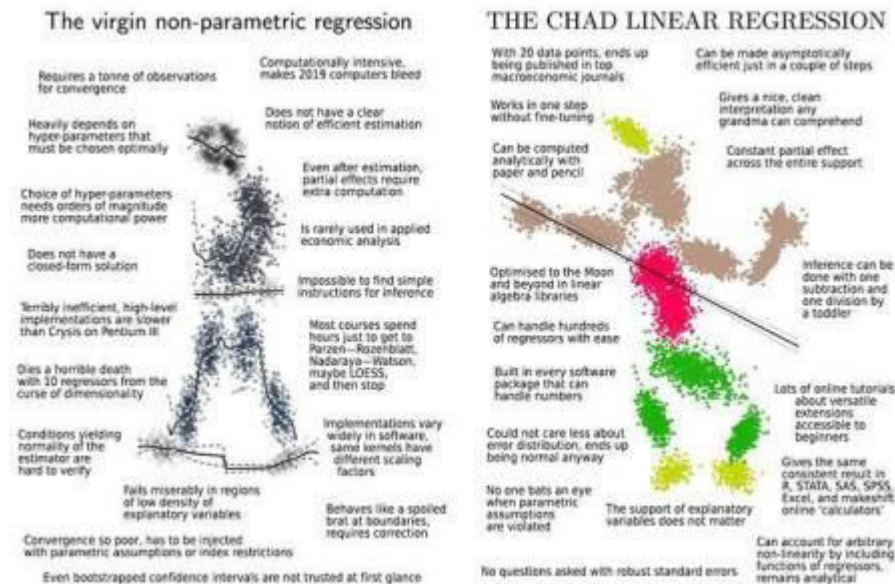
The features that are taken into account in the main models are:

- Date
- Indicator of home field advantage
- Teams skills
- Tournament importance (ex: Friendly Game rated as 1, Euro or World Cup Game rated as 8, etc.)
- Confederation of opponent
- Indicator showing if team is in list of "special playstyles" (composition of the list is selected to maximize prediction accuracy)
- Interaction term skill difference x tournament importance

Unfortunately the model does not take into account player-level data. For example, the absence of Milan AC's superstar [Zlatan Ibrahimovic](#) is not taken into account in Sweden's odds to win it all. Another great improvement could come from using [expected goals](#) instead of directly scored goals. The idea is that scored goals are a very small sample in soccer, so there's much more signal in xG. Unfortunately, computing these expected goals requires play-by-play data, which I don't have access to.

“I tried a lot of different methods, but none were able to beat my baseline set up with a simple linear regression”

As for the choice of the algorithm, although we had good success last time with splines, I also wanted to try various algorithms from the ML world in order to get the best possible results. In the end, I tried a lot of different methods, but none were able to beat my baseline set up with a simple polynomial regression. Perhaps unsurprisingly to many data science practitioners, linear regression wins again!



“The virgin non-parametric regression vs. The Chad linear regression” – meme taken from [Reddit](#)

This first part of the prediction pipeline returns a double (see image below), which unfortunately we can't directly use to create meaningful probabilities for the outcomes we're interested in (group stage games, group rankings, teams final rankings).

pred_atk	pred_def
1.5741378	0.9720111
1.0106497	1.3397924
1.0061946	1.6221507
1.2543437	1.1928407
1.3478684	1.1282584
1.5787923	1.3976596
2.0689635	0.6815113

Example of number goals predicted by the polynomial regression: results are floats, not integers

In order to get something useful, we have to resort to simulations. Simulations are really necessary in this case since a lot of non-linear factors can influence the final results: group composition, a somewhat [convoluted qualifying system](#), etc. To me these “nonlinearities” are the fun part though! For this reason, we'll update predictions after each day so that readers who are into this stuff can see how match results affect the likelihood of the global outcomes.

So, how do we transform a floating-point predicted number of goals into simulations? We start

by using the predicted goals as parameter of a [Poisson distribution](#). This generates all possible scores for each individual game and assigns a probability to each score. Then for each game we select at random one of the possible outcomes. This selection is done with unequal probabilities: the probability for each outcome to be selected corresponds to the probability indicated by the Poisson model (basically you don't want to end with as many 8-3s as 1-0s in your simulations!). This is done through functions available in the excellent package [sampling](#) (see illustration below).

```
129 # Select one "multiverse" result for all matches in group, according to probabilities given in
130 # "gen_all_scores_group"
131 draw_onesim <- function(df_sim_group) {
132
133   onesim_idx <- sampling::UPsystematic(df_sim_group$prob_Poisson_calibrated)
134   onesim_group <- df_sim_group[which(onesim_idx > 0),]
135   onesim_group
136 }
137 }
138 }
```

The main function used to select game outcomes at random according to probabilities generated by the Poisson process. It uses the method `UPsystematic` from the `sampling` package

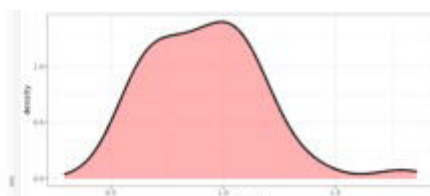
And then... we repeat the process. 10.000 times, Monte-Carlo style. With a little help from the great package [foreach](#) for easy parallelization of the computations. And voila, by counting and averaging the simulated results, we get probabilities for each match, each team occupying each rank in their group, and for each team to reach each stage in the second round. 😊

A small statistical twist

The Poisson distribution is often used in simulations for sports predictions, and it is generally a great tool to use. However, the raw results present flaws that can come bite you if you don't pay attention. For example, the Poisson tends to generate [far fewer draws](#) than expected (11% less in our case, to be precise). In order to account for this, we used one of my favorite tricks: [weight calibration](#) (implemented in the package [Icarus](#)). Basically, the idea is to modify the probabilities given by the Poisson distribution so that they match a certain pre-known result. There are infinite many ways to go about this, so weight calibration chooses the result that minimizes the distance to the initial probabilities. One of the other benefits of using this method is that we can calibrate on many probabilities at the same time, which we eventually did for:

- Total number of goals scored per match and average
- Share of draws
- Among draws, share of 1-1s, 2-2s and draws with more than 6 goals
- Share of draws in matches with "small" difference between teams skills
- Share of draws in matches with "large" difference between teams skills

Before calibration	After calibration	Range
0	20.80	19.7 - 20.7
1	39.18	37.8 - 39.8
2	25.11	25.8 - 25.8
3	24.75	22.7 - 22.7
4	16.42	12.8 - 16.8
5	5.28	6.8 - 6.8
6	1.36	6.8 - 6.8
7	2.13	2.8 - 2.8



Output from the calibration process given by the `Icarus` package

Model selection and results

In order to select the best possible model, we ran a whole lot of tests and hyperparameter selection scripts. The main metrics for the selection process were prediction accuracy and the [Brier score](#), which represents the quality of the estimated probabilities. Despite the simplicity of

the model, the accuracy of the overall prediction is ~61% +/- 2% (both on the test set and in Cross-Validation), which is [in line with most other soccer models](#). The 3-dimensional Brier score is 0.57 +/- 1%.