Open in app ↗ Sign up Sign In

Published in Towards Data Science

Jack Tattersall Follow

Jan 13, 2021 · 15 min read · Listen

Save

Using machine learning to identify high-value football transfer targets

Adventures with xgboost and the FIFA 20 dataset

62

1

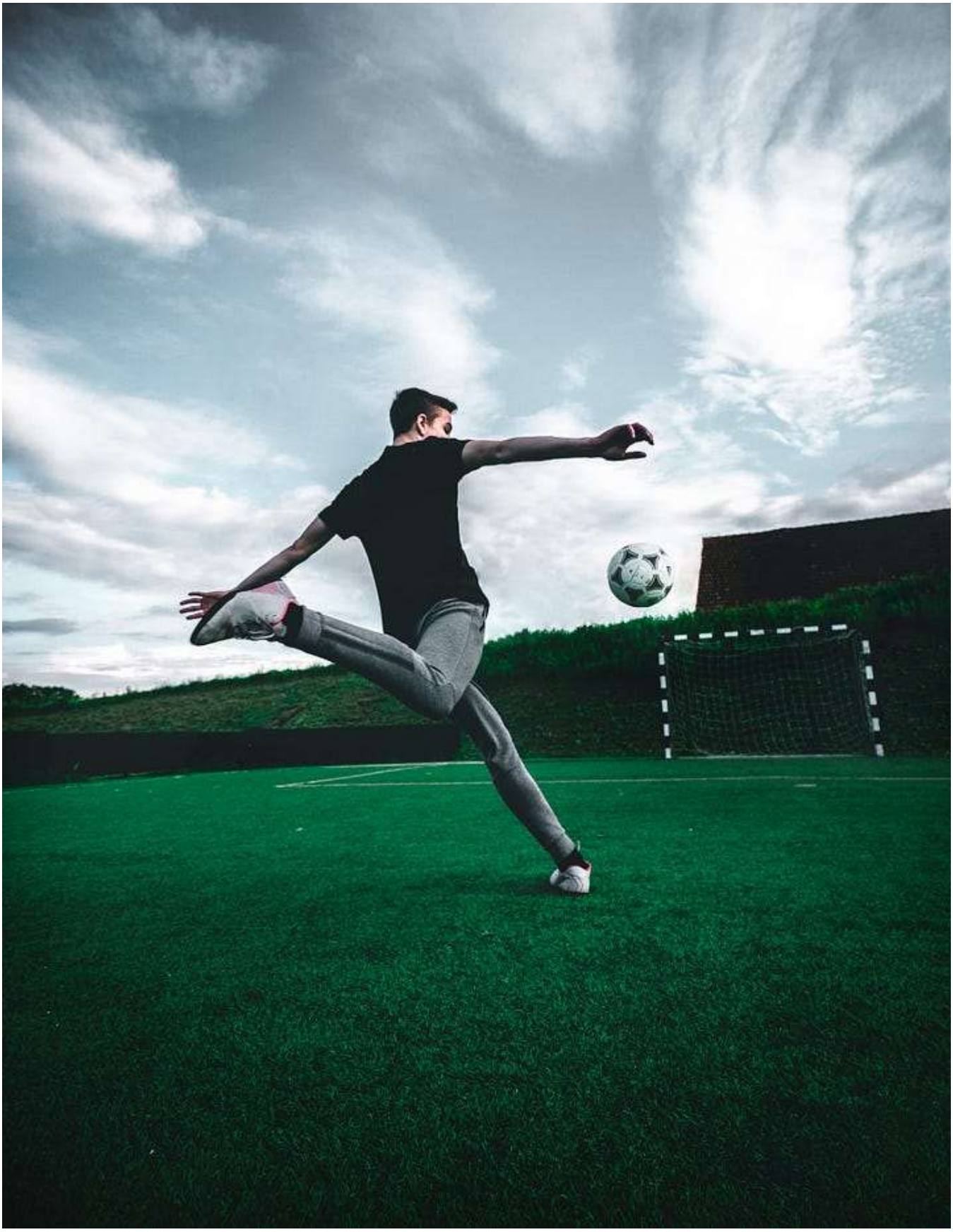

Photo by Sven Kucinic on Unsplash

If you want to replicate the analysis described in this article, you can find all of the code on GitHub here

Hindsight is a fickle thing.

It's all too easy to look back on events and think that we should've seen them coming. *Surely*, we think, we should have *known* that Bitcoin was going to be a thing, that Trump would win in 2016, that the iPod touch meant the end of phones with buttons on them.

The same is true of football transfers. It's hard not to look back on moves like Andy Carroll to Liverpool, Fernando Torres to Chelsea, Alexis Sanchez to United, and Andriy Shevchenko to Chelsea and think it must've been obvious that they were doomed from the start.

But reality is rarely that simple.

Let's take an example.

In the summers of 2016 and 2017 respectively, two 25-year old wingers moved to top Champions League sides for club-record fees. Both had been performing well at their old clubs; both were valued in the £20–30 million bracket; both had overall ratings in the low-80s on FIFA. One player had scored once every two games the previous season, the other once every three. They'd even both previously had a poor spell at a top-four Premier League club, before moving abroad to secure more game time.

So, what do you think? Should it have been obvious what the respective fortunes of these players was going to be? Do you think your answer would be different if you knew their names?

One of these two was Germany international André Schürrle. He moved to Borussia Dortmund in 2016 for a fee of around €30 million. Unfortunately for him, things did not work out at the Westfalenstadion. Schürrle played fifteen times in his first season under Tomas Tuchel, scoring just twice. His market value plummeted over the next twelve months, and three years after his big-money move he was out on loan at Spartak Moscow. He retired at the end of that season.

And the other one? Well, the other one was Mohammed Salah.

Machine learning and the question of bias

Although these might seem like inconsequential anecdotes, hindsight bias can have a bigger impact than you might expect. If we always look back and think that it should've been easy to sort a Schürrle from a Salah at the time, we will underestimate how hard it will be to do the same thing in future. And next time we're presented with a 25-year old winger scoring for fun in Europe who looks like the next big thing, we'll think it's *obvious* that they're a great transfer target. And we may inadvertently end up with more Schürrles than Salahs.

So, what can we do about it?

One thing we can do is to base our assessments on **more objective data**. If we can design and validate systems to learn patterns that are reliably associated with subsequent high performance, and prove that they work in the real world, we can start to make better decisions and take human bias out of the equation.

Of course, machines are not immune to bias — far from it. There are so many examples of biased algorithms from the last few years that it's almost become something of a cliché in the data science world. In particular, machine learning algorithms can often inadvertently *amplify* human biases, especially if trained on data generated by human judgments. This is often avoidable though: if you know where the biases can come from, you can work out how to eliminate them from your training procedure, and get better outcomes as a result.

But listen, I know you don't need me to tell you all this. You've seen Moneyball. You're here to read about how we can actually do it.

So let's crack on.

Coming up

In the rest of this post, we'll play with a toy example of how you can build a football transfer recommendation system using R, `xgboost` and the FIFA 20 dataset. Our goal here is to train an algorithm to **identify players who will perform well in the coming season**. Further, what we *really* want to know is: which players would likely perform well *at the club we support* if we signed them.

We'll structure this as a **supervised machine learning problem**, with a continuous target variable relating to future player performance, which we will aim to predict in advance.

The steps we'll take will be:

1. Get the data, engineer features and define the target variable
2. Exploratory visualisation
3. Train and validate a simple xgboost model
4. Create predictions on the latest data and explore the results

(If you're not interested in the code and just want some juicy transfer gossip, you can skip ahead straight to section 4)

Step 1: Loading the data & engineering features

First up, we'll load all the data files and do some basic cleaning. We'll also take the very detailed "position" feature from the FIFA data and turn it into a slightly broader set of categories (Goalkeeper, Centre Back, Full Back, Midfielder, Attacking Midfielder, Winger, Striker).

We also define our target variable here. At the start of each season, we want to predict how well each player will perform over the course of that season. In general, players that have a good season will have their overall rating increased by the developers in the next version of the game. Those that have a poor season will see their overall rating go down. So our target variable will be the **improvement** column, representing the **change in overall rating** from one season to the next.

Feature engineering

Next, we'll engineer some features that we think our machine learning algorithm will benefit from. Not all algorithms actually require this step, of course. If we were going down the deep learning route, then we could in theory specify a model complex enough to engineer its own underlying features, and learn from those. The problem is, that kind of approach requires a lot of data — probably more than we've got here. And remember: more complex doesn't always = better. High-quality engineered features that encode important domain expertise can substantially improve most machine learning models, and as football fans we likely have a surprising amount of useful knowledge that we can code into our model.

For example, we know that **football is a team game**. This means that the extent to which a player improves (or fails to) depends not only on their own skills and characteristics, but also those of their teammates.

These relationships are far from simple. On the one hand, you've probably heard pundits talk about how a player will benefit from having a better team around them. Would Joe Gomez have played as well as he did in 19/20 without Virgil Van Dijk next to him, and Alisson Becker in goal?

But on the other hand, having *too many* other good players in the team — especially if they play in the same position — is probably not so good from a development perspective, as it means you're less likely to get game time. You could make the case, for example, that Gabriel Jesus would have improved more at Man City if he wasn't constantly playing second fiddle to Sergio Aguero.

Using a tree-based model like xgboost can help us capture some of these nonlinearities. To give it something to work with, we'll create some columns for each player representing:

- The average overall rating of players at their club
- The average potential rating of players at their club
- The average overall rating of the *starting*/first-choice players
- Average overall rating of the starting attackers, midfielders, defenders and goalkeeper (as separate features)
- The average overall rating of *other* players at the club who play in the same position as the target player
- As above, but using the potential instead of overall rating
- As above, but the maximum rating instead of the average
- The player's ranking in the "pecking order" within their position (ordered by overall rating)

Note that we also one-hot encode our categorical variables.

Step 2. Exploratory visualization

Next, we can check out a few interesting features of our dataset. We find that our target variable has a slightly skewed, normal-ish distribution, with a mean of around +1:

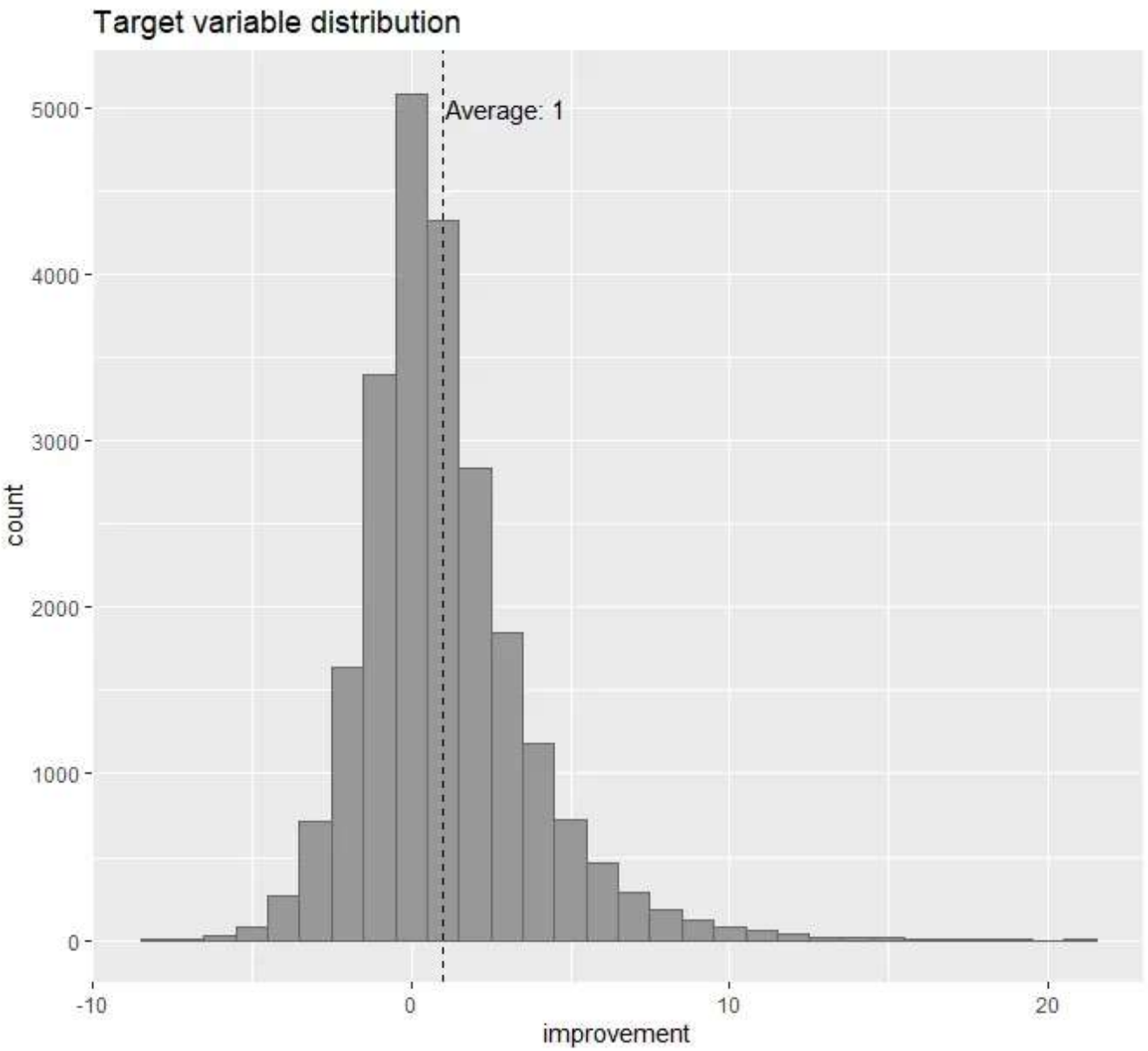

Image by author

We can plot the top 5 biggest winners and losers over the course of a season:

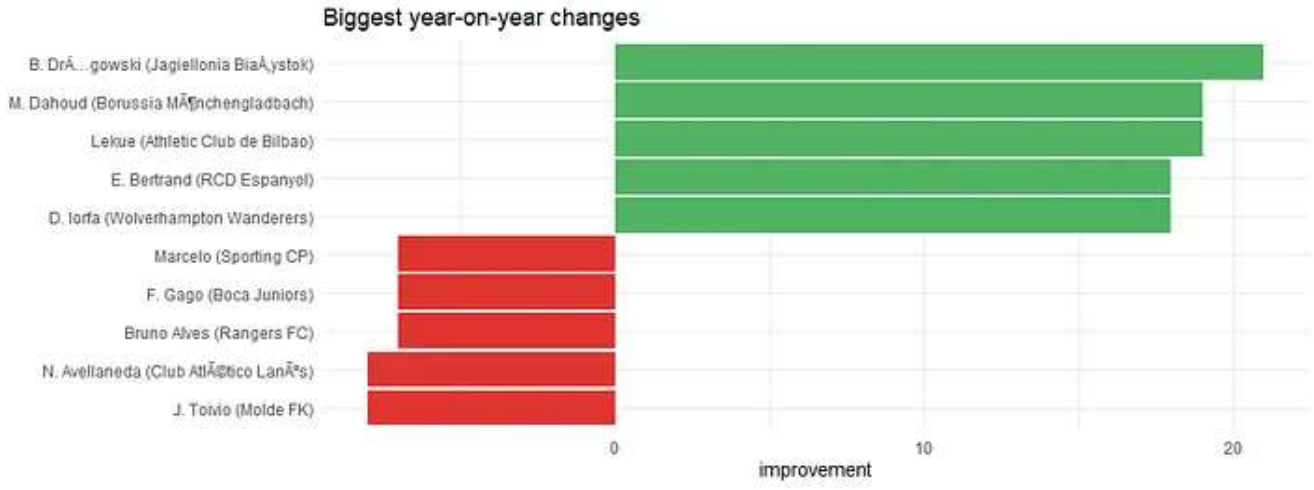

Image by author

The biggest improvement in the dataset was achieved by then-16 year old Polish goalkeeper Bartłomiej Drągowski, whose rating rose by a staggering *21 points* between 2015 and 2016, from 50 to 71. The biggest loser, on the other hand, was poor old Joona Toivio. The Finnish centre back's overall rating fell 8 points over the course of the 2018/19 season, from 71 down to 63. I don't know what he did wrong, but it can't have been good.

Intuitively, we might expect younger players to be more likely to improve from one season to the next, and older players to decline. It might also be the case that improvement tends to be faster in some positions than others, and there could even be an interaction between the two. Let's have a quick look at a plot of age and position vs improvement to see if anything jumps out. (Note we take a random sample of 500 players in each position to make plotting easier).

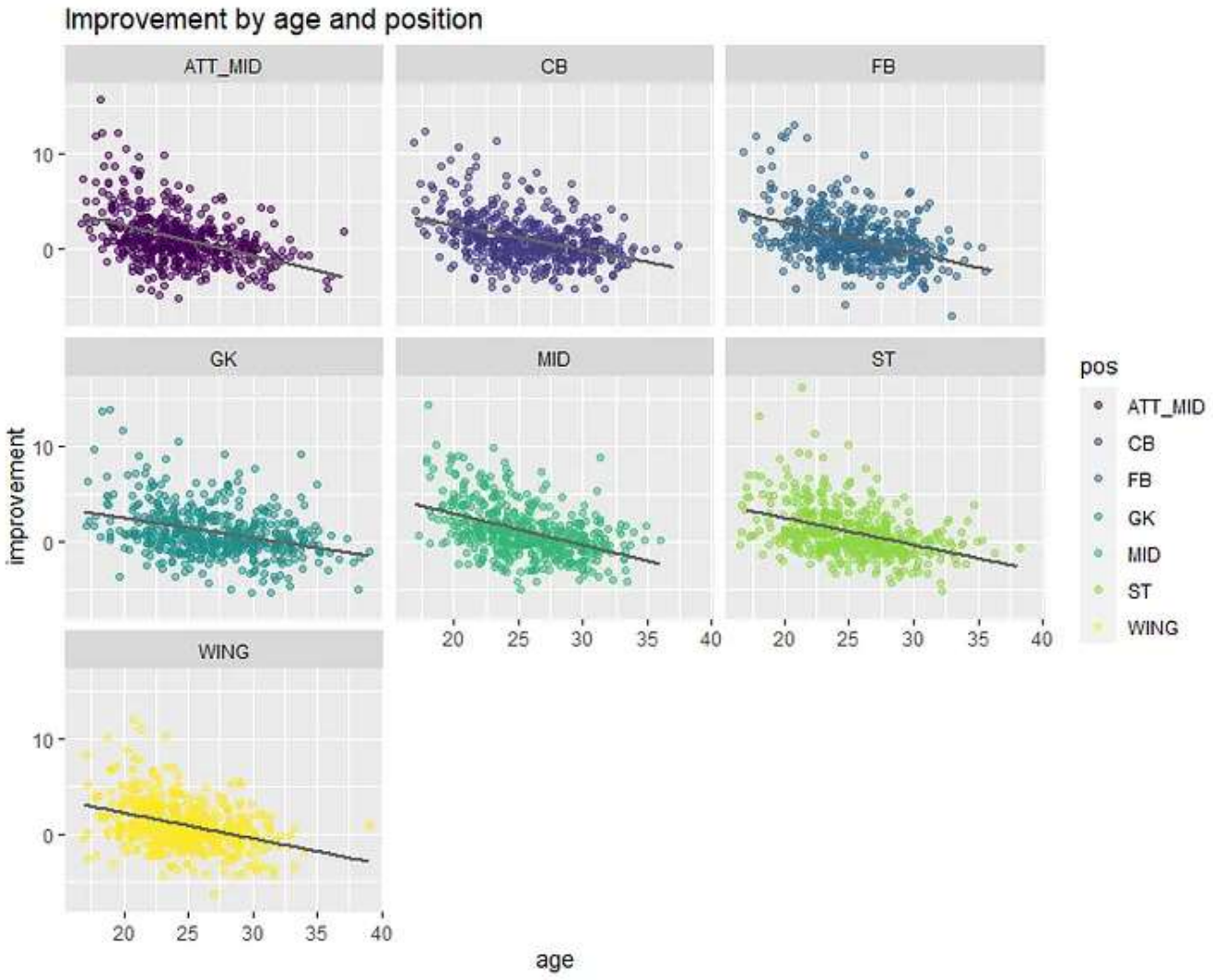

Image by author

As expected, there is an association between age and improvement — but there is also substantial variation around the average. There are plenty of younger players

who get worse from one year to the next, and plenty of slightly older players who get better. There are perhaps slightly more young attacking midfielders who improved in this data, and the relationship is possibly slightly shallower for goalkeepers than other positions, but otherwise the pattern is fairly stable across all groups.

Finally, let's generate some classic spider plots to look at those engineered team-level features.

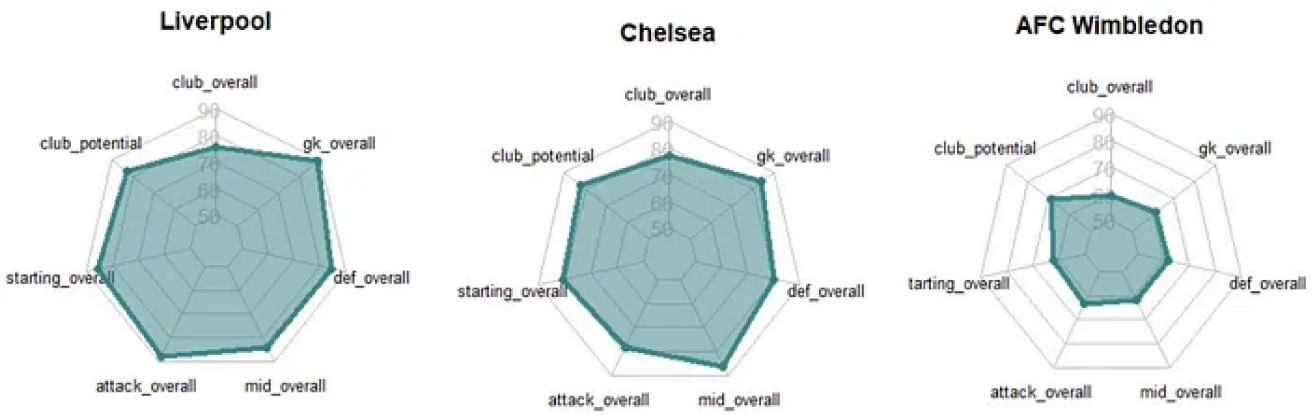

Image by author

The full code for creating these plots is below.

Step 3: Modelling and evaluation

Now that we've got our data and engineered our features, the actual code for the predictive modelling bit is pretty simple. Tuning, tweaking and improving the model would take time, of course, but for the purposes of this experiment some simple default parameters will do. All we have to do is split our data into train and test sets, convert both to xgb data matrices, and run them through the `xgb.cv` function. This will conduct k-fold cross-validation across our training set, which will give us estimates for how well we might expect this model to perform out of sample. By the way, this section assumes you're broadly familiar with xgboost. If you're not, there's a nice gentle introduction here you can check out.

One important thing to note here is that we are **not** splitting out our train and test sets randomly. If we did that, then we would be including cases in our training set from across the entire time period. This means that some of our training data would come from later in time than our test data. But the test set is supposed to represent "unseen" data that you wouldn't have access to in the real world until after you'd trained your model.

So instead, we replicate more realistic conditions, and split our data by time. This means that we will always be using past data to predict the future, and no time travel will be required.

We also add some code at the end to calculate a few different performance metrics for the best iteration, including RMSE and R squared. We might be interested in how well this performs as a basic improved/didn't improve classifier, so we add some classification metrics (sensitivity, specificity, positive predictive rate, and negative predictive rate) for good measure.

```
# A tibble: 1 x 8
model    rmse  r_square  sens   spec  pos_pred_rate neg_pred_rate
xgboost  2.15     0.339  0.332 0.915          0.651         0.741
```

The results we get look pretty reasonable, given that this is just xgboost straight out of the box. (Note that in the GitHub repo we also run a simple linear model as a baseline, which is always a good idea, but omit it here to save space). The RMSE score more or less suggests that the model gets its predictions right within about +/- 2 points on average, and the R square shows that we explain around 34% of the variance in the target variable.

For the classification metrics, the model correctly identifies around one third of the players that are going to improve, and correctly rules out around 90% of those that don't improve. Overall, if you used it to select a group of players predicted to get better in the coming season, around 65% of them would do.

Another important sense check is to look at which variables the model is using to make these predictions. If it's relying too heavily on just one feature, say, or on a set of variables that we wouldn't expect to be particularly important, then that could be a sign that something odd is going on. We can check this using the variable importance plot below (only the 20 most important features are plotted).

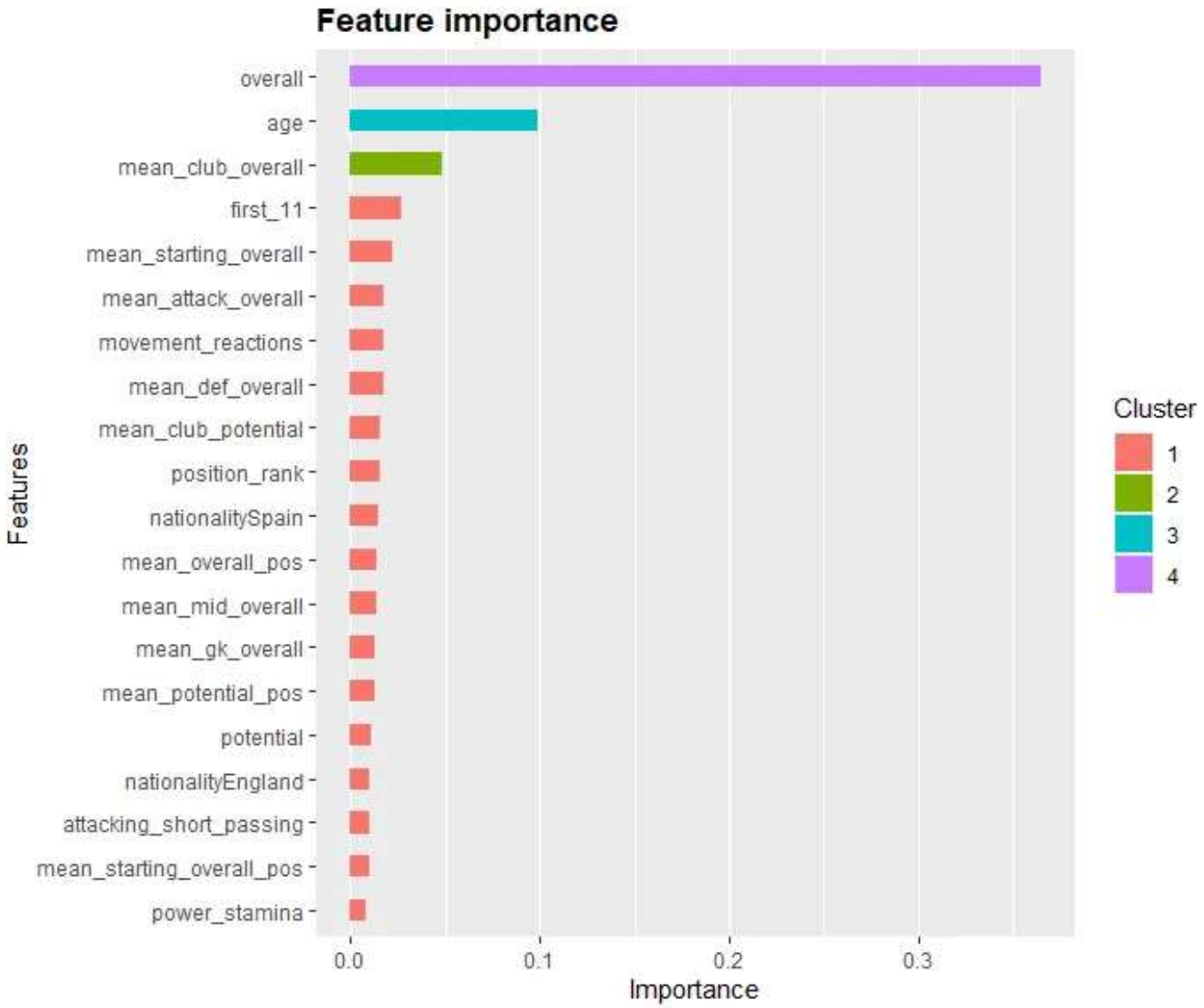

Image by author

It looks like the most important feature when predicting a player's future improvement is their current overall rating. That makes sense intuitively: players who are already very good are more likely to be close to the top of their game, and have less 'room' to improve into.

What's also promising is that the model is picking up effects for age, and for various features relating to the player's club (those extra columns we added earlier), including mean overall rating, mean starting 11 rating, position rank ("pecking order") and so on.

Finally, let's look at a few examples of player-level predictions. This is a random selection of predictions from the training data in cross-validation.

Note that our boys Mo Salah and André Schürrle are both in there — and that the model correctly flagged Mo as a good prospect (predicted to improve) and Schürrle as a poor one (predicted to get worse).

Since this model looks reasonable (and since this is just a toy example), we'll leave our evaluation there and accept it as our final model. We then test its performance on the hold-out data to see how well it does out of sample.

```
# Predict on test set
preds <- predict(xgb_model, newdata = test_xgb)

# Add to data
test <- test %>%
  mutate(pred_improvement = preds,
         residual = improvement - pred_improvement) %>%
  select(1:improvement, pred_improvement, residual, everything())

# Metrics
test %>%
  filter(season == 19) %>%
  mutate(pred = pred_improvement, obs = improvement) %>%
  mutate(pred_category = ifelse(pred > 0, "improve",
"decline/same"),
         obs_category = ifelse(obs > 0, "improve", "decline/same"))
%>%
  filter(!is.na(pred), !is.na(obs)) %>%
  summarise(rmse = RMSE(pred, obs),
            r_square = R2(pred, obs),
            mae = MAE(pred, obs),
            sens = sensitivity(factor(pred_category), reference =
factor(obs_category)),
            spec = specificity(factor(pred_category), reference =
factor(obs_category)),
            pos_pred_rate = mean(obs_category[pred_category ==
"improve"] == "improve"),
            neg_pred_rate = mean(obs_category[pred_category !=
"improve"] != "improve"),
            num = n()) %>%
  mutate(model = "xgboost") %>%
  select(model, everything())

## Results

model    rmse r_square   mae  sens  spec pos_pred_rate neg_pred_rate
xgboost  1.82    0.119  1.43 0.410 0.784         0.359         0.818
```

The model does reasonably well on the test data as well — better on some metrics vs cross-validation (RMSE, sensitivity, negative predictive rate), and worse on others (R square, specificity, postive predictive rate). There isn't any *major* evidence here for overfitting, so we're clear to try the model out for real, and start identifying transfer targets in the latest data.

Step 4: Identify transfer targets

Now we've got our model, we're ready to make predictions on the latest data and identify some transfer targets!

To do this, we'll set up a function that, for any given club:

1. Calculates the club's team-level attributes in the latest season
2. Creates a matrix of player ratings for the players at that club, by position. This is so that transfer prospects can easily be ranked against other players at the club in the same position, for creating some of the predictive features
3. Takes all of the players in the latest FIFA data, and sets their attributes so it "looks like" they're at the new club
4. Predicts their improvement score with these new features
5. Compares this with the original prediction (generated with the old club's features)

So essentially what we're doing here is saying: for every player in the dataset, how would their prediction change if their club-level attributes changed?

(We also add an option to get the *worst* transfers instead — i.e. transfer targets we should avoid, where we think a player would do worse at our club than if they stayed where they were).

If we run this for **Liverpool FC** and keep the top 100 targets, we get the results below. Some of the special characters have formatted a little…unusually, so you'll have to excuse those.

(*Note: when this post was created, the latest available data was the FIFA 20 dataset — so these predictions are for the 19/20 season and are a little out of date. Adding in the FIFA 21 data would be a good extension to the project!*)

Some interesting things jump out here:

- All of the top ten targets are attacking midfielders, including the likes of **Kevin De Bruyne** (we can dream), **Paulo Dybala** and **Bruno Fernandes**. Explaining the model's specific selections is something we'll probably save for a future post — but it looks like the model may be picking up on the relative lack of options available to Jurgen Klopp in the CAM department
- Several of these have actually been transfer targets for Liverpool in recent seasons — most notably **Nabil Fekir**. So it looks like the model is making at least some reasonable decisions!

- **De Bruyne** is a particularly interesting inclusion in the top spot. Although it's unlikely that transfer would ever happen, the model predicts that if he stays at Manchester City, KDB's overall rating will likely decline slightly in the near future — but that he'd actually *improve* if he moved to Anfield. Food for thought.

- Given the current defensive injury crisis on Merseyside, the selection of promising Leipzig centre back **Ibrahima Konaté** is an interesting one. The 25-year old has in fact been linked with a move to Liverpool in recent months, and our model predicts that he would **improve more than 60% faster** under Klopp than in his current club. And that's in a world where he's fourth in line behind Van Dijk, Gomez, and Matip — with more first-team exposure, he might be predicted to improve even faster.

We could probably spend all day poring over these results — and in fact this kind of manual inspection is often an important step in machine learning projects.

In the appendix below, you can see the top 50 transfer targets for every club that played in the Premier League in the 19/20 season. Which picks do you agree with for your club? Which ones do you think the model got wrong? Why might that be? Answering these kinds of questions can often help you identify new data sources to include, new features to engineer, more domain expertise to encode that might improve your future predictions.

Conclusions

There are of course a lot of limitations with the approach we've taken here. Although FIFA's ratings system is impressively thorough, it is naturally imperfect. Any message board discussing an individual player's ratings will be full of fans complaining about perceived inaccuracies, and a system designed for a videogame is never going to fully capture all the nuances of performance needed at a professional level. Our "improvement" target is therefore a little crude for similar reasons. There are also lots of more sophisticated ways that we could model the problem — e.g. using Graph Convolution Networks to more fully capture the complex interaction effects resulting from having different kinds of players at the same club.

However, what we've shown here is the tip of the iceberg. We've explored some simple, readily-available data, spun up a model in a few lines of code, and already obtained results that make intuitive sense. Even this simple proof of concept could

help spark some interesting transfer discussions. But top-level clubs have access to far richer and more extensive data sources than this. Just imagine what you could do if you had the resources to apply machine learning techniques like those explored here to something like the Wyscout database.

So the moral of the story is: when it comes to machine learning, start with something easy, find some easily-available data on a topic you're interested in, and don't be afraid to get stuck in and experiment. You might be surprised at what you find.

Appendix: All premier league transfer targets

The table below was generated by running the `get_best_transfers()` function for every team in the premier league in the 19/20 season. For each team, we use a couple of rough heuristics to filter to more "realistic" transfer targets (there's not point just recommending Lionel Messi to everyone).

Firstly, we filter to players who are recorded as having a market value equal to or less than the most expensive player in the target club + 20%, according to the FIFA game. Secondly, we also filter by overall rating, only showing players who are currently at most a couple of points worse on the "overall" rating than the worst player in the starting club's first 11. So if your team's worst starting player is rated as 80 overall, and your most expensive player is worth 50 million, you'll see transfer targets with current overall ratings of 78 or higher, and current market value of 60 million or less.

Search in the "new_club" column for the team you support, and have fun exploring the predictions!

Machine Learning Soccer Premier League R Data Visualization

Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. Take a look.

By signing up, you will create a Medium account if you don't already have one. Review our Privacy Policy for more information about our privacy practices.

Get this newsletter

About Help Terms Privacy

Get the Medium app

Download on the App Store

GET IT ON Google Play