Corners Challenge

Jean-Michel Alvarez

Language/IDE
Python was used (along with various libraries such as pandas, numpy, pyplot, xgboost, seaborn, sklearn, etc.) within VS Code for this challenge.

Leagues
The leagues were one-hot encoded as categorical variables as I saw some correlation between this feature and the total number of corners.

Datetime
I set a datetime index for each dataframe based on the Date column and then created `Year` (2005-2010) and `Day_of_year` (1-365) features to take into account variations in time. Not every team will perform the same in 2010 as they did in 2005, so I wanted to take that into account, especially considering the test data is so chronologically near the end of the training data, so this latter data should be given more importance.

Other features
The average goals and corners when at home and when away were calculated for each team and for each season. These values were then propagated through the relevant rows.
For example:
If team 410 scored an average of 1.3 goals per game when at home in 2005, then all rows with 410 in the `HomeTeamId` column and 2005 in the year received 1.3 in the `Home_Avg_Goals` column.
The same was done for average corners.

Target
Of course, a `Total_Corners` column was created which summed the corners in each game.

Preparing test data
The average goals and corners for each team were then pulled into the test dataframe from the training one. However, because training data from 2011 was not yet available, average values from the previous year (2010) were used instead. Various degrees of recency could be used, but I figured that taking the last year is likely to be more representative of a team's performance than their average performance over the last 5 years or, conversely, over the last week of 2010 for example.

Final processing
Several columns were then dropped from both the training and test dataset, resulting in dataframes containing only the features needed for our predictions. These were:

- One-hot encoded league IDs
- `Year`
- `Day_of_year`
- `Home_Avg_Goals`
- `Home_Avg_Corners`
- `Away_Avg_Goals`
- `Home_Avg_Corners`

Picking the right estimator
As we are attempting to predict a quantity with 23k data points and a relatively limited number of features, I knew this was a task for a not-overly complex regression model based on the Bias-variance tradeoff principle. Various immediately sprung to mind, and I went ahead and tested the following:

- XGBoostRegressor
- LassoCV
- ElasticNetCV
- RidgeCV
- SVR (linear)
- SVR (rbf)
- GradientBoostingRegressor
- RandomForestRegressor

From the very first run, most of these were resulting in R^2 values of around 0.1, which is not ideal. However, RandomForestRegressor (RFR) scored an impressive 0.85, much better than all the other models. It is possible that RFR's default settings were coincidentally more appropriate for the task than the others, and that had I spent more time individually tuning the hyperparameters of the other models I could have been able to achieve similar results, but nevertheless I decided to crack on with RFR.

Hyperparameter Tuning
5-fold RandomizedSearchCV and GridSearchCV were used to optimize various RFR parameters, such as `n_estimators`, `bootstrap`, and `max_depth`. Setting `bootstrap` to True and `max_depth` to 4 were clearly optimal, and as expected, accuracy went up with `n_estimators` (albeit with diminishing returns). I settled on 500 as a good tradeoff of computational time vs accuracy.
My tuned RFR had a final R^2 score of 0.87, which I worried seemed too good to be true, but the fact that I used 5-fold cross validation reassured me that the model was less likely to have overfit.

Predictions
The model was then used to generate predictions for the 341 test data points. These seemed reasonable from first glance, within 1 or 2 points from the `Line`, so I proceeded.

Calculating Probabilities using Poisson Distribution
I assumed a poisson distribution to calculate `P(Under)`, `P(At)`, and `P(Over)`. I evaluated the poisson distribution formula over each integer from 0 to `Line` and took the sum as `P(Under)`. If the `Line` was a half value, I set `P(At)` to 0, otherwise I calculated the probability of it being that exact discrete value. `P(Over)` was set to `1 - P(At) - P(Under)`.

Stakes
I used the Kelly Criterion to estimate how much I should stake on each bet. This initially resulted in a percentage of my total balance to stake on each bet, which did not take into account the results of previous bets nor my total balance. So I proceeded with the following assumptions:

- Do not assume bet outcomes
- All bets must be placed simultaneously
- Units cannot be divided

I only bet on games where the Kelly Criterion exceeded a minimum threshold. In this case, I used 0.25. Therefore I was only betting on games where according to Kelly I should stake over 25% of my total bankroll. I then scaled these stakes up to my total bankroll and rounded them. **This resulted in me placing 99 bets, with stakes ranging from 2 to 7 units, for a total of 340 units.**

Note that some games occurred in a league where I had no training data for (like league 811), I provided probabilities for these but avoided betting on them.