Soccer Betting

Param Shah, Zheyuan Hu, Gaomin Wu, Ruoxiao Yu, Toby Du
New York University, New York, NY 10003
prs392@nyu.edu, zh2095@nyu.edu, gw1107@nyu.edu, ry796@nyu.edu, yd951@nyu.edu

Abstract

Soccer betting is one of the most popular sports betting activities around the world. In this project, we explore whether we can make money consistently through soccer betting. Our project builds on the data scraped from four different websites. Advanced python techniques like multi-threading, Cython, and Numba are utilized to boost the performance of the scraper. Then we employ three feature engineering methods: Past Performance Indicators, Colley's Rating Method, and Neural Rating System, to evaluate performance of teams. Based on the features, a Logistic Regression Model producing binary classification results for whether the home team wins is built which achieves an out of sample accuracy of 65%. With the model, we then try out two betting strategies, either to bet on the predicted result only or to bet on a portfolio. Finally, a simplified simulation shows around 4% profit rate for a total of 1315 games.

1 Introduction

1.1 Background

Soccer, also known as football outside the US, is the most popular sport around the world with an estimated following of 4 billion fans. The popularity of soccer has aroused the development of soccer betting, and soccer's simple win-lose-draw outcome and low scoring also make betting more reasonable and meaningful.

Generally speaking, bookmakers provide odds of a match based on their prediction of the outcome and the money flow, which are obtained through mathematical models combined with the market. The buyers will then make their betting decisions on account of the odds given by the bookmakers as well as their own predictions of the match. In this project, we are going to explore if we can make profit in a stable way through soccer betting, that is, to beat the bookmaker.

1.2 Task Overview

In order to make money through soccer betting, we have to complete two tasks. First is to build a model that can accurately predict the outcome of a coming soccer match given the historical information. Second is to use some betting strategies to make our betting decisions, based on the predicted outcome obtained from the model and the odds offered by the bookmaker.

2 Data Preparation

2.1 Data Sources

The first step is to gather data to feed into the model. To make our data more accurate and to have more potential features, we decided to scrape data from four websites, and each site would give us different features we want. Table 1 lists the four source websites and the features to be scraped from the websites.

Source Websites	Features
fbref [1]	Match statistics
fivethirtyeight [2]	SPI ratings and rankings
understat [3]	Expected goals and forecasts
football-data [4]	Odds provided by bookmakers

Table 1: Source websites and corresponding features

2.2 Data Scraping

Knowing that scraping from websites is an I/O bound job and it usually takes a long time to run a serial-based scraper, our initial thought is that we can optimize the scraper with some advanced python techniques we learned in this class, like parallel computing or threading. Therefore, we first built some serial-based scrapers, and then optimized the scrapers with several different methods and compared their performance boost against the original one. Finally, we used the optimized scraper with the greatest performance boost to extract the full dataset.

2.2.1 Tune with Line Profiler

We started by testing the original serial-based web scraper on a single website to scrape the match reports for 200 games, which took around 14 minutes to run. This means that it will probably take days if we are going to use the serial-based web scraper to extract the full dataset, which includes 4 websites and information for 7000 matches.

By checking out the inside function with line profiler, we noticed that most of the time has been spent on accessing data from the website, and this is consistent with our initial thoughts that threading or parallel computing could help. Figure 2 in the Appendix shows part of the line profile results, where the total time is 815.18 s, and 40.3 % of time has been spent on accessing data from the website.

2.2.2 Advanced Python Techniques

Based on the original scraper, we tested the following three advanced python techniques:

- Concurrent multi-threading with 8 workers
- Compile with Cython and predefined ctype variables
- Parallel computing with Numba

Table 2 shows the tested results on the same 200 webpages. Threading has the greatest performance boosting, which is around 600 % speed up compared to the original scraper. Therefore, we decided to implement threading on the original scrapers we built to extract the full dataset.

Method	Runtime	Speed Up
Serial-based (original)	815.18 s	0 %
Multi-threading	137.54 s	592 %
Cython	425.04 s	191 %
Numba	364.49 s	223 %

Table 2: Performance boosting for each method

Finally, we made use of decorators based disk caching (using diskcache) to cache the downloaded webpages since we were aware of the fact that we would be blocked several times by the website due to several requests in a short time. Doing this avoided re-accessing the same webpage that has already been cached.

2.2.3 Data Preprocessing

After scraping, we obtained four raw datasets, each of which corresponds to one source website. In order to get a single dataset that is ready for feature engineering and modeling, we did the following preprocessing procedures sequentially

- Aggregate player statistics to team level
- Merge these datasets by bringing all the data to the same format (for team names, leagues, seasons, dates, etc.)
- Treat missing values
- Drop useless columns
- Convert non-numerical features to numerical dtypes

One advanced python technique applied here was, making use of numba's Just in time compilation for calculation of levenshtein ratio for linking team names from one dataset to another dataset. It only provided a little speed up (881 ms ± 68 ms per loop to 742 ms ± 37.9 ms per loop).

The final dataset contains all matches in the Big 5 European Leagues (English Premier League, Spanish La Liga, German Bundesliga, Italian Seria A, and French Ligue 1) from 2017 to 2021, 6974 matches in total with 76 features each. Figure 3 in the Appendix shows what the dataset looks like.

3 Feature Engineering

We perform three kinds of feature engineering for this project. The first one deals with making use of performance indicators such as goal differentials, shot differentials, etc. For the second one, we make use of the Colley's team rating system [5]. Finally, we propose, design and test our own team Offensive and Defensive rating system.

3.1 Past Performance Indicators

Most of these variables are the difference in average feature differentials in the past games. For example, consider two teams A and B playing against each other. We wish to calculate past performance indicators using the feature f. The way we would generate a feature using f to predict who will win the current game, is by calculating the average difference of f between team A and all the other teams that team A has played against in the past, making a similar calculation for team B, and taking the difference between these two values.

This is something that could be done in a loop, by iterating over each game and all its past games. But, as we have learnt in class, we should make use of `numpy` and `pandas` to perform vectorized operations. We used `wide_to_long` in `pandas` to transform the data from "per game" to "per team"

while keeping track of which game each row belongs to. Then, we used `pandas` group by, rolling mean, and shift to calculate the features for each team. Afterwards, the dataset was rejoined to bring it back to the "per game" format, and finally the difference was calculated.

To assess how fast using vectorized operations of pandas are compared to looping, we wrote a function that calculated all the past performance features using loops and another function that used pandas features. We found (using %%timeit in jupyter) that looping took about 3.75 s ± 129 ms per loop and using pandas took 2.56 s ± 55.6 ms per loop.

3.2 Colley's Rating Method

Colley's rating method uses results from the past games in the season to generate an estimate of ratings of a given team in a particular league.

The Colley's method uses the following formula for generating the rating of team i:

$$r_i = \frac{1 + (w_i - l_i)/2 + \sum_{j \in O_i} r_j}{2 + t_i}$$

where w_i is the number of games won by team i before the current game, w_j is the number of games lost by team i before the current game, t_i is the total games played by team i before the current game, and O_j are the list of opponents that team i has faced in that season.

One important thing to note here is that, the unknown team i's rating depends on the unknown team ratings of team j which i has faced. Hence it is important to convert this problem to a form that is easily solvable. By performing some linear algebra, the above equation can be formulated as a linear algebra problem:

$$\mathbf{C}r = b$$

Where r are the unknown rating vectors for the n teams (therefore has a shape of n x 1). b and $\mathbf{C}$ are defined as follows:

$$\mathbf{C}_{ij} = \begin{cases} 2 + t_i & i = j \\ -n_{ij} & i \neq j \end{cases}$$

$$b_i = 1 + \frac{1}{2}(w_i - l_i)$$

We solved this linear system using Ordinary Least Squares approach. There is one major optimization possible in this process. Since the ratings generally change after every game, we need to re-perform the OLS calculation after each game. Since we already have the data, we could make use of multiprocessing to speed up this operation. We could not implement this due to time constraints.

3.3 Neural Rating System

Neural rating system is a method devised by us to rate the offense and defense strength of a team based on several features. The major motivation to perform this was that we were able to extract good quality features of every single player in every match in the Big 5 leagues since 2017. It begged the question of how we could leverage these player level features to describe the strength of one team when it is playing against another.

Our idea was developed from the "Soccer Ratings - An Optimization Approach" blog post [6]. The main approach of this article was to make use of Mean Squared Error to regress the product of home team's offensive rating with the away team's defensive ratings to the goal scored by the home team, while regressing the product of Away Team's defensive rating with the home team's offensive ratings to the goal scored by the away team at the same time. We followed a similar approach with some adjustments.

First, we realized that most of the offensive rating of a team is dependent on the attackers and the midfielders of the team, so we aggregated the collected features per match for only these two groups of players for each team. Let's call the aggregated features of these offensive players for a particular match $\overrightarrow{f_O}$. Similarly, only the midfielders and the defenders are generally responsible for the defense of the team and are the ones who drive the defensive rating, so we aggregate features of these defensive players for a particular match, and call them $\overrightarrow{f_D}$.

Second, the offensive rating of a team for a particular match is a function of $\overrightarrow{f_O}$, let's say $g_O(\overrightarrow{f_O})$ and the defensive rating is a function of $\overrightarrow{f_D}$, say $g_D(\overrightarrow{f_D})$. To find these functions, we make use of neural networks. These neural networks are 2-layered, making use of Linear Layers, trainable Batch Norm and ReLU as the non-linearity. We also apply ReLU just before the output because we wanted to constrain the ratings to be positive only. The loss function we used is as follows:

$$loss = \sum_{(h,a)} (G_h - (g_{O_h}(\overrightarrow{f_{O_h}}) * g_{D_a}(\overrightarrow{f_{D_a}})))^2$$

$$+(G_a - (g_{O_a}(\overrightarrow{f_{O_a}}) * g_{D_h}(\overrightarrow{f_{D_h}})))^2$$

Where h is the home team and a is the away team, and G_h and G_a are the goals scored by home and away teams when playing against each other.

The ratings generated by this model were interpreted with respect to manual market research and the standings of the team in the leagues.

4 Predictive Modeling

To predict the game outcome of whether the home team wins or not, we made use of a simple Logistic Regression Model. From the "Predictive Analytics with Sports Data" course, we learned that logistic regression tends to outperform other models with this kind of data. Also, it can provide the significance of each feature. We made use of the `statsmodel` package in python. The data used to train and validate the model was from 2017-2018 and 2018-2019 season and the out of sample accuracy was calculated on the 2019-2020 season. The out of sample accuracy achieved with only past performance indicators was 63%. The accuracy we achieved using only the differential in colley ratings as the feature is: 62.55%. The final features chosen contained the Neural Ratings, the SPI ratings and the game importances obtained from the FiveThirtyEight dataset. We were able to achieve an out of sample accuracy of 65% compared to the random accuracy of 50%. We also made a multiclass model using the multinomial logit model in `statsmodel` to try and predict the outcome as home win vs away win vs draw. In this, we achieved an out of sample accuracy of 51% compared to the random accuracy of 33%.

5 Betting Strategy

We test the ML model we have to simulate the betting in Season 2019-2020. In our betting strategies, we use both predicted probability from the ML model and the public odds information. The odds tells us how much we will get back if we bet \$1, and it reflects probability from the bookie. For example, a home win odds of 1.8 reflects a probability from the bookie of 55% (1/1.8) for home win.The rule is simple: we have \$100 budget for each game while betting, and we want to see the net profit we achieve after betting the whole season with two betting strategies.

5.1 Betting Strategy I - One Bet

The first betting strategy is to bet on the predicted result only. For each match, a prediction will be made by the ML model. We will make one bet only and bet on the predicted result. With the prediction, we use the highest odds among 7 online betting houses, which means if we win, we get the highest profit possible. This will be the odds at which we place the bet. For each match, the amount of bet will be calculated by the Kelly criterion [7], which works based on the principle: you should invest only a fraction of your wealth. By keeping some aside, one will not end up in bankruptcy. The optimal fraction (f) depends on each individual bet:

$$f = \frac{Edges}{Odds} = \frac{p_w^* x - (1 - p_w^*)}{x}$$

$$p_w^* = \frac{1}{\frac{1}{n}\sum_1^n x_i}$$

where x_i is all the odds in the market.

Specifically, we aggregate all odds from many different betting houses to get a better reflection of how bookmakers view the probability of an event.

5.2 Betting Strategy II - A Portfolio

With the second strategy, the bettor benefits from spreading the bets over multiple opportunities in one round. For each game with a fixed budget of \$100, we allocate some fraction of the fixed budget B_i to all three results: Home, Draw and Away. And a value bet indicator will tell us when to bet. We will only bet when the predicted probability is greater than the probability reflected from the odds that bookies provided.The fraction B_i is calculated below:

$$b_i = \frac{B_i \mathbf{1}\left(\hat{p_i} - 1/o_i\right)}{\sum_j^3 B_j \mathbf{1}\left(\hat{p_j} - 1/o_j\right)}$$

where $\mathbf{1}$ is the indicator function evaluating to 1 for positive arguments, and to 0 otherwise. B_i is a hyper-parameter, $\hat{p_i}$ is predicted probability from ML, and $1/o_i$ is probability reflected from odds.

There are actually four variants of this betting strategy, each with different B_i. The four strategies spread the bets on such opportunities:

- uniformly, i.e., $B_i \equiv 1$
- according to the estimated probability of winning, i.e., $B_i = \hat{p_i}$
- according to the absolute difference between win probabilities predicted by the model and that of the bookmaker, i.e., $B_i = \hat{p_i} - 1/o_i$

- as above, but using relative difference, i.e., $B_i = (\hat{p}_i - 1/o_i)/\hat{p}_i$

We will refer to these strategies as unif, conf, abs-disc, and rel-disc, respectively.

5.3 Simulating Result

Simulation	Profit
sim1	-25.5
sim2 unif	4924.0
sim2 conf	2869.7
sim2 abs-disc	1365.1
sim2 rel-disc	3345.0

Table 3: Net profit of each simulation

We simulated on Strategy I and the four variants of Strategy II, and got the cumulative net profit throughout the 2019-2020 season as shown in Table 3. We can see that with a total of 1315 games, and $100 budget for each game, the best net profit of $4924 is achieved by Strategy II with uniform spread of bets. Figure 1 visualizes the cumulative net profit of the strategies.

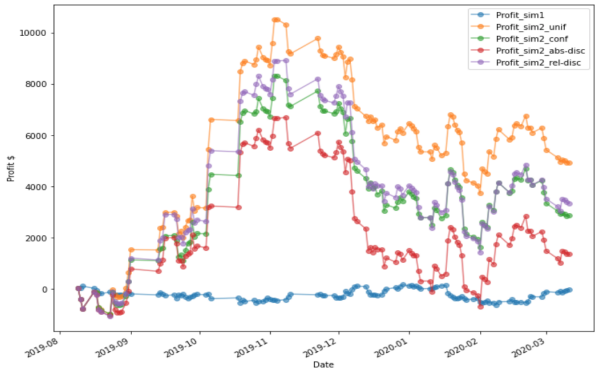

Figure 1: Cumulative net profit for 5 simulations

References

[1] https://fbref.com/en/comps.

[2] https://data.fivethirtyeight.com/.

[3] https://understat.com/.

[4] https://www.football-data.co.uk/data.php.

[5] Wesley N. Colley. Colley's bias free college football ranking method: The colley matrix explained, Sep 2018.

[6] Kpele. Soccer ratings: An optimization approach, Jun 2017.

[7] J. Kelly. A new interpretation of information rate. *IEEE Transactions on Information Theory*, 2(3):185–189, 1956.

6 Appendix

6.1 Line Profiler

```
Total time: 815.18 s
File: <ipython-input-4-415dae2f7cd4>
Function: get_df at line 1

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     1                                           def get_df(urls):
     2                                               '''
     3                                               Description: This function goes to de URL of the m
     4
     5                                               Input:
     6                                                   - urls: urls of all the match reports
     7                                                   - league: league
     8                                                   - season: season
     9                                               Output:
    10                                                   - Dataframe treated from the match saved on my
    11
    12                                               '''
    13         1       5895.0   5895.0      0.0      df_all = pd.DataFrame()
    14         1         11.0     11.0      0.0      ind = 0
    15
    16       381       3500.0      9.2      0.0      for url in urls:
    17                                                   # make the request
    18       380       2152.0      5.7      0.0          pg = 'https://fbref.com'
    19       380       3667.0      9.7      0.0          url_pg = pg+ url
    20       380 1015519715.0 2672420.3     12.5           req = requests.get(url_pg)
    21       380       9102.0     24.0      0.0          if req.status_code == 200:
    22       380      33529.0     88.2      0.0              content = req.content
    23                                                   # accessing data from site
    24       380 3282314181.0 8637668.9     40.3           soup = BeautifulSoup(content, 'html.parser')
```

Figure 2: Part of line profile

6.2 Final Dataset

	Team_Home	Team_Away	USxg_Home	USxg_Away	Date	USProb_Home	USProb_Away	Season	League	Goals_Home	...	FBREFSCA_Away
0	Monaco	Toulouse	2.146360	0.248798	2017-08-04	0.8672	0.0223	2017-2018	Ligue1	3	...	11.0
1	Paris S-G	Amiens	2.252640	0.190301	2017-08-05	0.8981	0.0144	2017-2018	Ligue1	2	...	8.0
2	Montpellier	Caen	1.461790	0.319240	2017-08-05	0.7392	0.0547	2017-2018	Ligue1	1	...	12.0
3	Lyon	Strasbourg	1.727830	0.292445	2017-08-05	0.8120	0.0327	2017-2018	Ligue1	4	...	5.0
4	Saint-Étienne	Nice	0.943684	0.593773	2017-08-05	0.4413	0.1732	2017-2018	Ligue1	1	...	10.0
...	...	...	...	...	...	...	...	...	...	...	...	...
6969	Huesca	Getafe	0.472579	0.711933	2021-04-25	0.1954	0.3926	2020-2021	LaLiga	0	...	9.0
6970	Nice	Montpellier	2.376730	1.730330	2021-04-25	0.5245	0.2430	2020-2021	Ligue1	3	...	24.0
6971	Wolves	Burnley	0.351673	2.240490	2021-04-25	0.0220	0.8866	2020-2021	EPL	0	...	19.0
6972	Benevento	Udinese	2.518620	1.349530	2021-04-25	0.6746	0.1379	2020-2021	SerieA	2	...	14.0
6973	Fiorentina	Juventus	1.211380	1.156030	2021-04-25	0.3491	0.3046	2020-2021	SerieA	1	...	16.0

6974 rows × 76 columns

Figure 3: Final dataset