CS230

European Soccer League Outcome Predictor

Sang Ahn
Graduate School of Business
Stanford University

Stefan Elbl Droguett
Graduate School of Business
Stanford University

Saman Tabatabaee
School of Medicine
Stanford University

Abstract

We build a predictor for the result of soccer matches in the Top 5 European Leagues. We present different models that vary in complexity and track their accuracy. Introducing player level data marginally improves the accuracy compared to our baseline model. We identify draws as the main factor that reduces the accuracy of our results and explain our rather unsuccessful attempts to remedy this problem.

1 Introduction

Towards the end of the 2021-2022 season, Liverpool FC was two games away from achieving an unprecedented feat: winning the quadruple . They were already champions of England's two domestic cups, and were two wins away from the English league and the Champions League, Europe's most prestigious club title. However, luck was not on their side, and what was once a team that instilled fear in their rivals is now at the middle of their domestic table, not qualifying for any European cups this tournament and with a performance that has shocked the experts who considered them a strong contender for all of the competitions they were in at the start of the year. Had one asked any football fan on the street about their expected performance for this coming year, they would have most likely stated their preference for them. How could have anyone predicted the disaster they are living in now? The previous case serves as the motivation for our project: **Creating a predictor of football matches**.

To carry our task, we consider team level data and player characteristics for each team as inputs. The former refers to composite scores such as ELO[1], how well the team has performed in previous matches[2], offensive and defensive forms[3] and the pre-game betting odds from Bet365. At the player-level, we use data from the FIFA videogames for each respective season to characterize each player with 34 features that track offensive and defensive performance. Each of these features corresponds to a score between 1 and 99 (higher is better) and is updated biannually. Examples of those features are: short passing accuracy, sprint speed, acceleration, balance and shot power. The output of our model is a prediction of whether the match will end as a win, draw, or loss for the home team.

Initially, we construct a baseline model that only compares the ELO of both teams through a soft regression, which is used as a reference of comparison for our main model. Afterwards, we add player level data to construct a fully connected neural network with 5 hidden layers, each with leaky

[1]Competitive score metric that tracks how "good" a team is compared to their peers
[2]For simplicity, we focus on how many wins, draws and losses they had in their previous 5 matches
[3]How many goals they have scored/received in the last matches

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

ReLU activation functions and an output layer with softmax. We apply dropout for regularization and mini-batch gradient descent for computational efficiency. We also utilize Adam optimizer instead of standard gradient descent.

2 Related work

Given the popularity of soccer and the potential profitability of accurate predictions in sports betting, there have been previous attempts to predict football match outcomes using machine/deep learning methods. For instance, Hukalijuk and Rakipovic [1] use data from the UEFA Champions League to predict match outcomes and compare the performance of various learning algorithms. Out of Naive Bayes, Bayesian Networks, LogitBoost, KNN, Random Forest (RF), and ANN, they conclude that ANN performs best. Timmaraju et al. [2] use data from the English Premier League (EPL) and focus on feature selection rather than learning algorithms. Their results suggest that using temporal gradient k-past performances (TGKPP) to choose features improves prediction accuracy. In another report by Rodriguez and Pinto [11], feature selection based on Support Vector Machines (SVM) seems to give the highest accuracy prediction (61%) for EPL data. However, other methods such as ANN and RF produce a general accuracy between 50% and 60% accuracy, with better clustering of wins and losses (50% - 90%) and struggling in predicting draws (<30%). Similarly, Baboota and Kaur [4] explore various combinations of feature engineering methods and learning algorithms using data from the EPL. They conclude that gradient boosting, a learning algorithm that performs feature selection by itself, shows best performance.

In general, the state-of-the-art approach for soccer match prediction seems to come from more clustering techniques than neural network approaches, with an emphasis on finding features that can describe the variance in the outcome. In contrast to these papers, Shin and Gasparyan [3] use virtual data from a video game (FIFA 15) to predict actual match results in the Spanish La Liga with K-means clustering. They find that the performance is comparable and sometimes even better than predictions based on actual data. The incorporation of explicit player data and its higher accuracy (>70%) motivated us to apply a neural network approach to this input format for our main model. Previously, Nivetha et al. [5] use EPL data and apply an RNN-based approach to predict match outcomes. Specifically, they use LSTM (long short-term memory) and demonstrate that it outperforms traditional machine learning algorithms and ANNs. Since player data increases the dimensions of the input vector, we decided to use a traditional ANN model, and if successful, then expand it to an RNN-approach for future work.

3 Dataset and Features

We mainly rely on data obtained from Kaggle. The "European Soccer Database" from Kaggle includes data from top-tier soccer leagues in eleven European countries (Belgium, England, France, Germany, Italy, the Netherlands, Poland, Portugal, Scotland, Spain, and Switzerland) from the 2008/2009 season to the 2015/2016 season. It has data on approximately 25,000 matches, 300 teams that took part in those matches, and 10,000 players who played on those teams. At the match level, it tells us the number of goals scored by each team and the players who started the game. It also has pre-match odds for home team win, draw, and home team loss from various betting companies, but we only keep the odds from Bet365 which has the largest coverage. This part of the data is based on information from online sports betting websites. At the player level, it contains 34 features for each player which are related to the player's various skills such as passing, offense, and defense. The player attributes are obtained from EA's FIFA video games, who provide biannual updates on these metrics based on actual results in the real world.

We restrict our attention to the big 5 leagues (England, France, Germany, Italy, and Spain) because they receive the most extensive media coverage and thus they are more likely to have full and accurate information. Also, big leagues are systematically different from smaller leagues due to their spending power and the presence of superstar players. After discarding the smaller leagues and matches with insufficient information, we are left with 12,633 matches. We use 73.64% of the sample (9304 matches) as the training set (08/09-13/14 season), 13.38% (1690 matches) as the development set (14/15 season), and 12.97% (1639 matches) as the test set (15/16 season). Figure 2 in the Appendix shows the number of matches by season in our final data set.

We complement the Kaggle data with ELO scores from clubelo.com. ELO ratings are a method for calculating relative skill levels in competitions. It was originally developed for use in chess but has been used extensively in other sports such as soccer, baseball, and basketball. The difference in ELO scores between teams serve as a predictor for the outcome of the match; the greater the difference, the higher the probability that the high-ELO team will win. After each game, the winner takes ELO points from the loser and a team can gain more points by winning against opponents with higher ELO ratings. Thus, a team that is under-rated or over-rated in terms of ELO quickly converges back to a rating that reflects their true level.

Figure 3 in the Appendix shows descriptive statistics for team-level features. Home (away) team match form is based on the home (away) team's results in the last five home (away) games. We award 1/0/-1 points to wins/draws/losses and compute the sum. For example, a match with a home team that has won four out of the past five home games and lost the other would have a home team match form of 3. Home (away) team offensive form is the sum of goals scored by the home (away) team in the last five home (away) games. Home (away) team defensive form is the sum of goals conceded by the home (away) team in the last five home (away) games. B365H/B365D/B365A are pre-match betting odds from Bet365 and respectively indicate the odds for a home team win/draw/home team loss(=away team win). In our analysis, we take the inverse of the odds to make them more like probabilities.

Figure 4 in the Appendix shows the list of player features that we use. Although there are actually 748 player level features for each match with 34 features per player, 11 players per team, and 2 teams per match, we take the average of player features for each team to reduce the number of features and increase processing speed. Thus, we have 68 player features per match with 34 for home team and 34 for away team. For brevity, we do not include the descriptive statistics for player features. In sum, for each match, we have 79 features (11 team level features and 68 player level features).

4 Methods

For the loss function, we use the standard categorical cross-entrophy loss function for soft-max problems: $\mathcal{L} = -\sum_{r\in\{W,D,L\}} y_r log(\hat{y}_r)$, where r is each of the possible results from the perspective of the home team: win, draw, or loss. We decided on this function since we are concerned with higher accuracy. The function punishes the algorithm every time it mislabels an example.

Our output layer uses a soft-max activation function with 3 labels. It takes output of the last hidden layer (L_k, dimension $m \times n_k$ with n_k being the number of neurons in last hidden layer), and returns the predicted probabilities of a home team win, draw, and a home team loss. We randomly generate a weight matrix (W, dimension $n_k \times 3$) and a bias vector (b, dimension 1×3, addition below carried out with broadcasting) from a multivariate standard normal distribution and multiply them by a small number to reduce their magnitude. Then we calculate the probability of each match for being a win, draw or loss. Formally:

$$Z = L_k W + b \, , \sigma(Z_i) = \mathbb{P}[\text{Result being i}] = \frac{e^{z_i}}{\sum_{r\in\{W,D,L\}} e^{z_r}}$$

We then introduce 5 hidden layers with leaky ReLU activation functions. Each layer has weight matrices W_1, W_2, W_3, W_4, W_5 and bias vectors b_1, b_2, b_3, b_4, b_5. The leaky ReLU function takes the form $Z = \max\{\eta z, z\}$, with η being a hyper-parameter. We use leaky ReLU as opposed to standard ReLU because it never has zero slope and thus could help speed up learning and prevent problems with vanishing gradients.

For the final output, we obtain a $(m, 3)$ vector of probabilities. For each training examples, we have a (1,3) probability vector that sums to one. We identify whichever label among win, loss, and draw has the highest probability take that label as the predicted label.

Assuming we are using a standard gradient descent, our strategy for back-propagation would be as follows. In the output layer, $dW = \frac{1}{m} L_5' (\sigma(Z) - Y)$ and $db = \frac{1}{m}\Sigma(\sigma(Z) - Y)$ where L_5 is the output of the fifth hidden layer , Y is the one-hot encoded version of the true label vector, and Σ is summation within the same column. For dL_5, we take $dL_5 = (\sigma(Z) - Y)W'$ and multiply an element by η whenever an element with the same index in L_5 is less than 0 (non-linearity of leaky ReLU). dW, db, and dL_5 have the same dimensions as W, b, and L_5. In the fifth hidden layer, $dW_5 = \frac{1}{m} L_4' dL_5$ and $db_5 = \frac{1}{m}\Sigma dL_5$. For dL_4, we take $dL_4 = dL_5 W_5'$ and multiply an element by

η whenever the element with the same index in L_4 is less than 0. We repeat the process for the fifth hidden layer for earlier layers until we obtain dL_1, dW_1, and b_1. Finally, we update the Ws and bs according to $W = W - \alpha dW$ and $b = b - \alpha db$ where α is a hyper-parameter that represents the learning rate.

However, for more efficient updating of gradients, we rely the ADAM algorithm instead of standard gradient descent in our model. For ADAM we define four additional auxiliary variables: V_{dw}, V_{db}, S_{dw} and S_{db}. All of them are initialized to zero. They are updated according to the following:

$$V_{dw} = \beta_1 V_{dw} + (1 - \beta_1) dW \text{ , } V_{db} = \beta_1 V_{db} + (1 - \beta_1) db$$
$$S_{dw} = \beta_2 S_{dw} + (1 - \beta_2) dW \text{ , } S_{db} = \beta_2 S_{db} + (1 - \beta_2) db$$

V corresponds to the "momentum" of the gradients and S corresponds to the RMS-Prop component. For each epoch they are "corrected" from the distortion the mini-batch could create according to: $V_{di}^{\text{Corrected}} = \frac{V_{di}}{1-\beta_1^t}$ and $S_{di}^{\text{Corrected}} = \frac{S_{di}}{1-\beta_2^t}$ with $i \in \{W, b\}$. Finally, W and b are updated according to:

$$W = W - \alpha \frac{V_{dw}^{\text{Corrected}}}{\sqrt{S_{dw}^{\text{Corrected}}} + \epsilon} \text{ , } b = b - \alpha \frac{V_{db}^{\text{Corrected}}}{\sqrt{S_{db}^{\text{Corrected}}} + \epsilon}$$

with ϵ being a small positive number.

5 Experiments/Results/Discussion

In order to obtain the best performance from our algorithm, we performed tests over different sets of hyper-parameters. Our criteria was to prioritize those combinations that yielded higher accuracy (our primary evaluating metric) without taking too much time to train and run the model. Our key hyper-parameter was the learning rate for ADAM. Lower learning rate led to lower oscillation around the convergence point (higher accuracy) but also resulted in slower convergence which would require us to increase the number of epochs. Thus, we experimented randomly among a host of numbers on log-scale to find the optimal learning-rate. Other parameters for ADAM did not seem to have much impact on accuracy or processing time, so we just used the default values from Tensorflow Keras. Similarly, η (leaky ReLU parameter) did not seem to have much effect either. We found that having more than 5 layers or 100 neurons per layer decreased processing speed without improving performance. Overfitting didn't seem to be much of an issue in our data and increasing the dropout rate seemed to result in slower convergence. Having a mini-batch size that is too high or low relative to the full batch size (9,304 training examples) led to longer processing time.

Finally, we chose the following set of hyper-parameters: 5 hidden layers with 64 neurons each, learning rate of 10^{-6}, β_1 of 0.9, β_2 of 0.999, ϵ of 10^{-7}, η of 0.1, drop-out rate of 0.2, mini-bath size of 256, and 10,000 epochs. In Table 1 below, we tabulate the accuracy of a baseline model (which uses just ELO scores as input features and relies on softmax with no hidden layers) and the main model for each of the data sets. In Figure 1 below, we show the confusion matrix. In Figure 5 in the Appendix, we plot how the value of the loss function and accuracy evolve over epochs.

Data Set	Acc Baseline (%)	Acc Main (%)
Train	46.92	51.95
Dev	47.63	53.20
Test	45.62	50.46

Table 1: Accuracy of our algorithm for Each Data Set and Model

Unsurprisingly, the main model outperforms the baseline model. However, Table 1 indicates the improvement over the baseline model is not as large as we expected. The confusion matrix in Figure 1 reveals that our model has difficulty in predicting draws. We hypothesize that this is because the nature of draws is much more random than that of wins and losses. Furthermore, if we consider the accuracy of our model on the test set conditional on the match not being a draw, we obtain a 68.18% accuracy, which lends credit to the idea that our model works in most occasions and that there is an inherent component in draws which is lowering our accuracy.

In order to remedy this problem, we create alternative models that deal with draws specifically. To reduce training time, we use a learning rate of 10^{-5} and 1,000 epochs in the subsequent models.

		predicted label			
		win	draw	loss	subtotal
true label	win	602	0	123	725
	draw	313	0	113	426
	loss	263	0	225	488
	subtotal	1178	0	461	1639

Figure 1: Confusion Matrix

In our first supplementary test, we train a second model that predicts draws versus non-draws by relabelling wins and losses into non-draws. We modify the output of our main model when the second model predictor predicts a draw. However, the predictor only returns non-draws, presumably because draws occur so randomly and the fully connected neural network cannot find a way to learn to predict them. The evolution of loss and accuracy for this model over epochs are plotted in Figure 6 in the Appendix.

In our second supplementary test, we train two models: a third model for predicting losses versus non-losses (relabel home team wins and draws into non-losses) and a fourth model for predicting wins versus non-wins (relabel home team losses and draws into non-wins). We change the output of the original model into a draw only when the third model predicts a non-loss and the fourth model predicts a non-win. The evolution of loss and accuracy for these model over epochs are plotted in Figure 7 and Figure 8 in the Appendix. While this increases the instances of draws in our prediction, it does not increase accuracy significantly. Based on the figures, the evolution of the main model is most similar to the one of the home loss versus non-loss model. This seems to suggest that non-losses, which include both draws and wins, and pure wins share similar characteristics and confuse the model. We conjecture that this is the reason our model fails to predict draws and loses accuracy.

6 Conclusion/Future Work

We develop a soccer game result predictor. Using a database of matches in the top 5 European Leagues between the 2008/2009 and 2015/2016 seasons, our model reaches a 50.46% accuracy in our test set, which is greater than the 33.34% of pure chance. Our model shows that including data at the player level increases the accuracy in comparison to using the team's ELO score as a composite metric, albeit not by a significant amount. Our results validate the usage of player-level data to predict soccer match outcomes and provide a promising venue.

The main shortcoming of our model is that it fails to predict draws. This problem is not unique to our model and has been a recurrent issue in the literature on soccer match prediction. For instance, Baboota and Kaur [4] use linear SVM (support vector machines) to predict soccer match outcomes and also find that the algorithm fails to predict draws. When we exclude draws from our test sample, our model reaches 68.18% accuracy. We provide alternative models to account for this phenomenon, and while one of them yields draw predictions, it does not recognize draws in a sufficiently robust manner as to increase accuracy. A venue for future work would be to develop better algorithms that better predicts draws. Our analysis indicates that draws seem to be more random in their nature relative to losses and wins.

7 Contributions

While all of the members contributed in each of the project's different tasks, there was one member in charge of each different task: Sang was in charge of data collection and developing the main model with player data, Stefan was in charge of the baseline model and writing, and Saman was in charge of data visualization and literature review.

References

[1] Hucaljuk, J., & Rakipović, A. (2011, May). Predicting football scores using machine learning techniques. In 2011 Proceedings of the 34th International Convention MIPRO (pp. 1623-1627). IEEE.

[2] Timmaraju, A. S., Palnitkar, A., & Khanna, V., "Game ON! Predicting English Premier League Match Outcomes"

[3] Shin, J. & Gasparyan, R., "A novel way to Soccer Match Prediction".

[4] Baboota, R., & Kaur, H. (2019). Predictive analysis and modeling football results using machine learning approach for English Premier League. International Journal of Forecasting, 35(2), 741-755.

[5] Nivetha, S. K., Geetha, M., Suganthe, R. C., Prabakaran, R. M., Madhuvanan, S., & Sameer, A. M. (2022, January). A Deep Learning Framework for Football Match Prediction. In 2022 International Conference on Computer Communication and Informatics (ICCCI) (pp. 1-7). IEEE.

[6] Mathien, H. (2016) European Soccer Database. *Kaggle* **Link**

[7] Chollet, F. & others, (2015). Keras. Available at: **https://github.com/fchollet/keras.**

[8] McKinney, W. & others, (2010). Data structures for statistical computing in python. In Proceedings of the 9th Python in Science Conference. pp. 51–56.

[9] Harris, C.R. et al., 2020. Array programming with NumPy. Nature, 585, pp.357–362.

[10] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[11] Rodrigues, F., Pinto, Â., (2022) Prediction of football match results with Machine Learning. Procedia Computer Science, 204, 463-470.

Appendix

Data Description

season	# of matches	%
2008/2009	1071	8.48
2009/2010	1521	12.04
2010/2011	1644	13.01
2011/2012	1659	13.13
2012/2013	1696	13.43
2013/2014	1713	13.56
2014/2015	1690	13.38
2015/2016	1639	12.97
Total	12633	100

Figure 2: Data Composition by Season

Variable	N	Mean	Std Dev	Minimum	Lower Quartile	Median	Upper Quartile	Maximum
Home_Elo	12633	1690.44	110.87	1423.82	1615.78	1673.63	1745.56	2106.16
Away_Elo	12633	1691.21	112.18	1426.49	1615.30	1674.56	1747.09	2107.48
Home_Team_Match_Form	12633	0.97	2.14	-5.00	-1.00	1.00	3.00	5.00
Away_Team_Match_Form	12633	0.84	2.15	-5.00	-1.00	1.00	2.00	5.00
Home_Team_Offensive_Form	12633	7.76	3.51	0.00	5.00	7.00	10.00	29.00
Away_Team_Offensive_Form	12633	5.77	2.93	0.00	4.00	5.00	7.00	24.00
Home_Team_Defensive_Form	12633	5.63	2.67	0.00	4.00	5.00	7.00	18.00
Away_Team_Defensive_Form	12633	7.61	3.11	0.00	5.00	7.00	10.00	22.00
B365H	12633	2.61	1.79	1.04	1.67	2.10	2.80	26.00
B365D	12633	3.82	1.17	1.40	3.25	3.40	3.80	17.00
B365A	12633	4.69	3.78	1.08	2.60	3.50	5.25	51.00

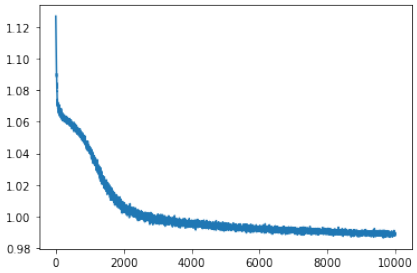
Figure 3: Descriptive Statistics - Team Level Features

overall_rating	long_passing	stamina	standing_tackle
crossing	ball_control	strength	sliding_tackle
finishing	acceleration	long_shots	gk_diving
heading_accuracy	sprint_speed	aggression	gk_handling
short_passing	agility	interceptions	gk_kicking
volleys	reactions	positioning	gk_positioning
dribbling	balance	vision	gk_reflexes
curve	shot_power	penalties	
free_kick_accuracy	jumping	marking	

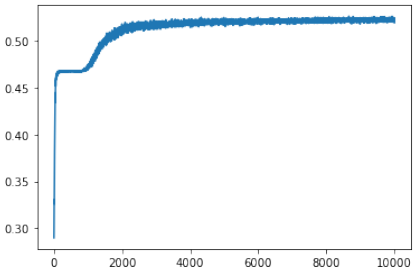
Figure 4: List of Player Level Features

Loss and Accuracies

Figures 5 - 8 represent the evolution of both the loss function and the model's accuracy with respect to the epochs in which they were calculated. In all figures, the x-axis corresponds to the epochs and the y-axis to the corresponding metric.

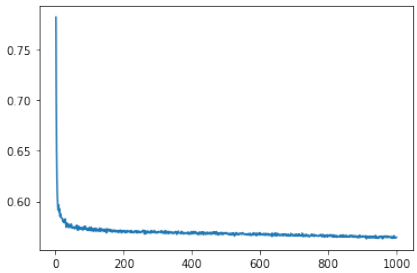


(a) Value of Loss Function

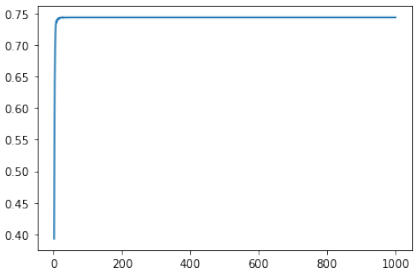


(b) Accuracy

Figure 5: Evolution of Loss and Accuracy for Main Model

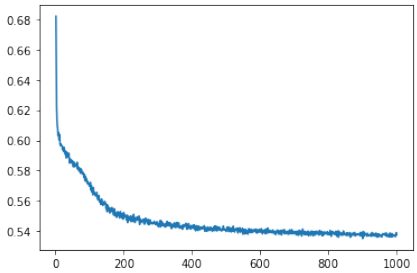


(a) Value of Loss Function

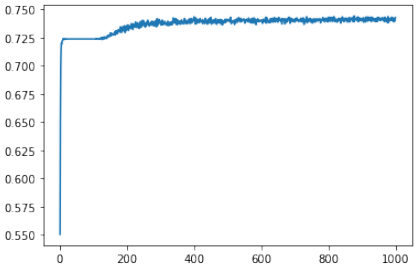


(b) Accuracy

Figure 6: Evolution of Loss and Accuracy for Draw vs. Non-Draw Model

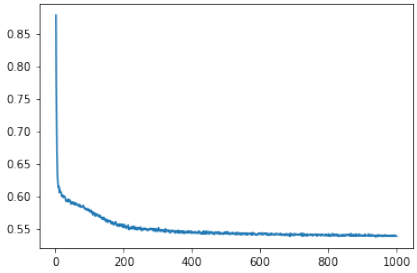


(a) Value of Loss Function

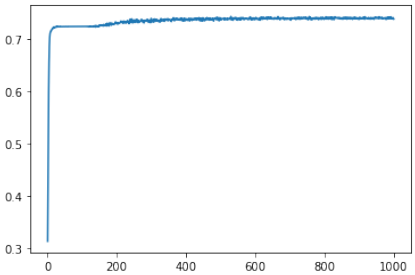


(b) Accuracy

Figure 7: Evolution of Loss and Accuracy for Loss vs. Non-Loss Model



(a) Value of Loss Function



(b) Accuracy

Figure 8: Evolution of Loss and Accuracy for Win vs. Non-Win Model