DEEP LEARNING, SPRING SEMESTER 2020

FOOTBALL PREDICTION - PREMIER LEAGUE

GROUP 04

Abdallah Zaher, M20190684
Cristina Mousinho, M20190303
Gabriel Ravi Santos, M20190925

Table of contents

Introduction

Football is one of the biggest sports in the world, if not the biggest. Some of the most famous players such as Cristiano Ronaldo, Lionel Messi and many others have a huge influence on the new generation around the whole world. Kids play it on the streets, stadiums, or school grounds. It doesn't matter if you're poor or rich, football will unite people for the love of this sport. A lot of them have their love grow up with them, even if they realize they aren't actually the next best player in the world. After playing and watching so many games, so many teams, so many leagues, their brain starts recognizing the patterns that lead to a victory. So as they get older, as their love gets bigger, so does their knowledge. Just in 2017, Placard Apostas Desportivas, the biggest betting game in Portugal profited over 457 millions of euros.

Many have tried to build Machine Learning models in order to predict results. And that's a team we want to take part of. That's exactly what led us to try and build a model that does just as well, if not better, than a long-term football lover.

The problem

As you probably already know, there are many types of bets one can pursue. From final scores to which and when players will score.

To keep things simple, we began by looking to predict the final outcome (home team wins, loses, or draws). This of course makes our problem a multi classification one. You will later see that we had to change our problem to a binary one, this time trying to predict if the home team simply wins or loses.

We decided to restrain our problem only to the english Premier League, as it is the biggest in the world.

The data

Data sources

To model our problem, we used mostly the UK's historical results, which are available online. There are files for every season since 1993/1994. Because football players play on average for 8 years, we decided to use data only from 2004/2005 up. Because the teams change so frequently, going too far back in history could turn on us. The file for each season comes with a lot of information, most of it about bets and game information. Unfortunately, those variables won't be used by us. We don't want other algorithms built by other companies influencing ours, so all betting data was dropped. Also, we want to be able to predict a match before it happens, so all data that relates to the games also has to be dropped (red cards, corners, goals, etc).

Though we lost a lot of variables from the original set, you will later see we managed to build new ones.

The other data source was data weather. It is true that the UK has quite a stable rainy climate, but we weren't sure if that could influence outcomes, so we did add weather data. We couldn't find data for specific dates for free, but we did find the monthly averages for the necessary years. The data came associated with a weather station, so we had to cross the home team's stadium location and station location to find the data we were looking for. In the end, we were able to do it pretty smoothly.

Data preparation

Fortunately for us, the football dataset was already clean. We found a couple of rows with no data, so it was just a matter of dropping them. It was here we found our dataset to be imbalanced, which later caught up to us while trying to model the multiclass problem.

As for the weather, some data was indeed missing, and we filled it using pandas' back-propagation filling option.

New variables

As we said before, a lot of variables had to be dropped, so we had to focus on the history of teams. That being said, we built the following variables:

- Cumulative goal difference over the season and over the past 3 weeks (if a team loses by 2 and then wins by 3, by the third game, their cumulative score is 1);
- Cumulative points over the season (everytime a team wins they accumulate 3 points, everytime a team draws they accumulate 1 point, and everytime a team wins, they accumulate 0 points).

We also extract month and week from the dates. From the week of the year, we managed to also obtain the week of the season (the first week there's a game being week 1). As a result and because we got the data per week (round), we dropped the first 3 rounds from our dataset for the reason that the players are still not in their best fit and they're back into the field after a long break and we need to be as accurate as possible.

In the end, we had historic information on the teams' performance over the season, information on the match (teams playing, referee, date, etc) and weather information (total rainfall over the month, average maximum temperature over the month, average minimum temperature over the month, and days of air frost over the month).

Methodology

Evaluation metric and protocol

Since we don't have a variance in the cost between false negatives and false positives, the chosen evaluation metric was accuracy. Since our goal is to predict the future based on history, we want to make sure our data maintains its sequence. Because of that, the chosen evaluation protocol was a simple hold-out validation. We actually did two different approaches, one with splits 80%/10%/10% and another with splits 90%/5%/5% for training, validation and test, respectively. In the first,

- Seasons from 2005/2006 to 2015/2016 were used for training;
- Seasons 2016/2017 and 2017/2018 were used for validation;
- Seasons 2018/2019 and 2019/2020 were used for testing.

And in the second,

- Seasons from 2005/2006 to 2017/2018 were used for training;
- Season 2018/2019 was used for validation;
- Season 2019/2020 was used for testing.

Dense Neural Networks

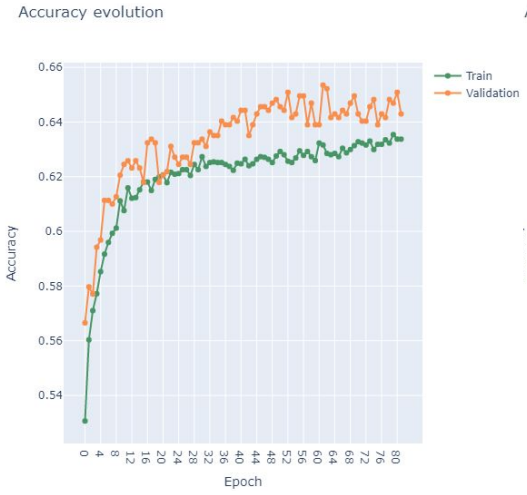
We started the modelling by encoding our variables and checking their importance using the RFE approach. In the beginning, we tried predicting if the home team lost, won or drew (so, a multiclass problem), but our results were incredibly low, not passing an accuracy of 0.55.

To the left, we have one of those first models. In it, three hidden layers were used. The first having 64 neurons and activation ‘relu’, the second having also 64 neurons and the same activation function, and the final one having activation ‘softmax’.

All these models overfitted, and we seemed to be getting nowhere. So we tried the second method, this time predicting only if the home team wins or not.

For this second approach, all variables were put to use, and we started by implementing a simple 2 layer one with activation ‘relu’, and a second with activation ‘sigmoid’. For this model, the first split approach was used (80/10/10), and so was the optimizer ‘adam’. As for batch size, it was 128. In terms of epochs, we used 200, but also early stopping (with patience 20), so the model actually never reached the 200 epochs.

Firstly, we explored how changing the size of the hidden layers affected the performance of the model.

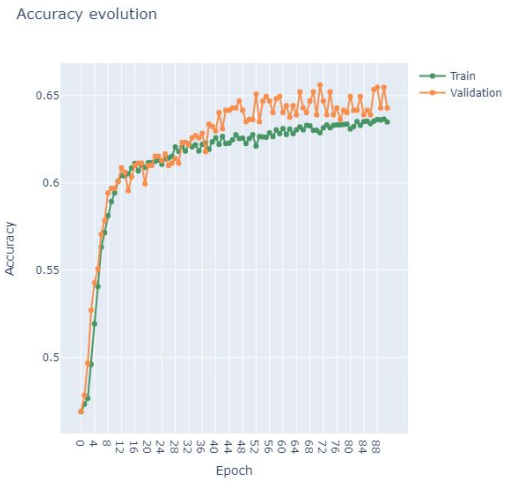


Hidden layer size: 8

Max. validation set acc.: 0.6535

Hidden layer size: 16

Max. validation set acc.: 0.6455



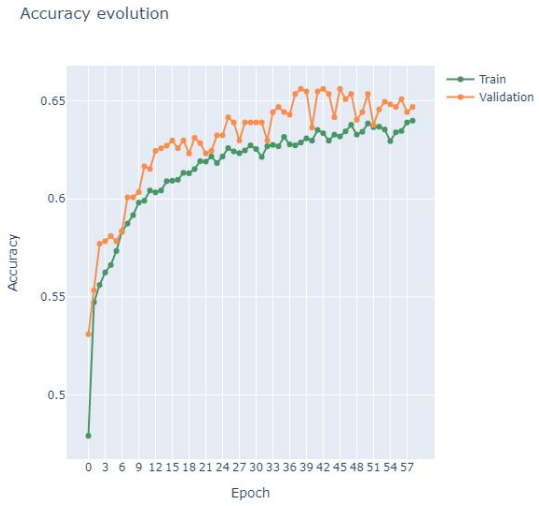
Hidden layer size: 12

Max. validation set acc.: 0.6561

Then, we explored the impact of the activation of that first layer. Here are the 2 best results we got.

Activation: ‘relu’

Max. validation set acc.: 0.6561



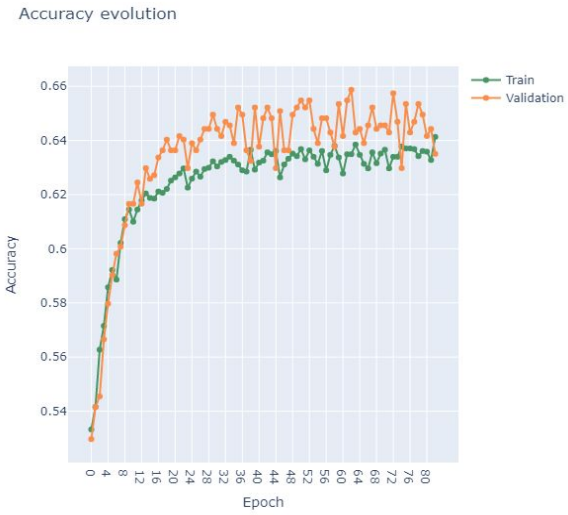
Activation: ‘exponential’

Max. validation set acc.: 0.6561

We moved on to exploring different types of optimizers, the following being our 2 best results.

Optimizer: ‘adam’

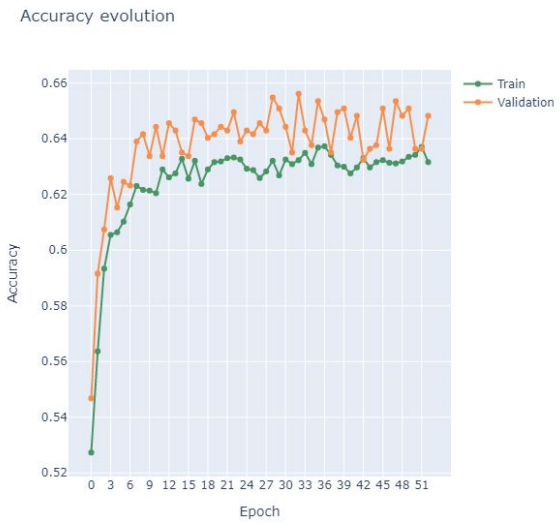
Max. validation set acc.: 0.6561

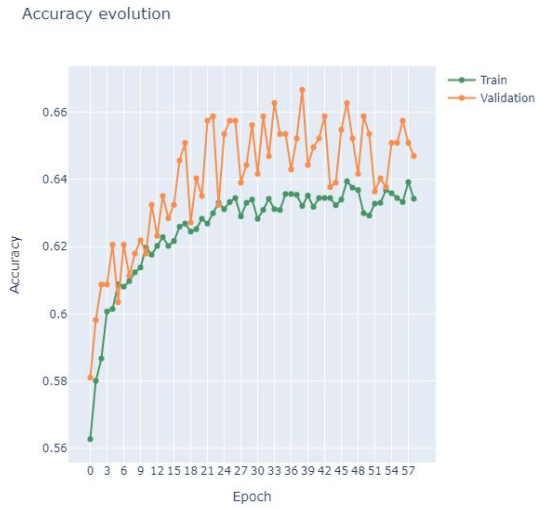


Optimizer: ‘Nadam’

Max. validation set acc.: 0.6588

After that, we tried adding layers with different sizes, but saw no improvement. Even though we got a result just as good, keeping the two layer option seemed like the right thing to do, as it resulted in less running time. We then turned to the kernel initializer of our layer. These where the best:





Kernel initializer: ‘glorot_uniform’

Max. validation set acc.: 0.6561

Kernel initializer: ‘orthogonal’

Max. validation set acc.: 0.6667

Finally, we played around with different regularizers, both l1 and l2, but once again saw no improvement. That being said, with this first model, the best accuracy we reached was 0.6667.

Long Short-Term Memory Neural Networks

Now that we reached an acceptable accuracy with DNN, we want to maintain information in 'memory' over time and we will apply LSTM Neural networks to check if we can find a better accuracy.

We generated more than 20 combinations of LSTM parameters and with a multiclass label (Win, Draw and Lose) but the accuracies were below 0.3; in this report we will only show the results that have a better prediction. The split approach that was applieds is 90%/5%/5% for training, validation and test, respectively.

At first, we applied ‘sigmoid’ and ‘tahn’ as activation functions, two layers, no Dropout and multiclass label:

Optimizer: ‘tahn’

Max. validation set acc.: 0.4988

Optimizer: ‘sigmoid’

Max. validation set acc.: 0.2584

After some studies, we noticed that we needed to change the activation function,label strategy (transform to binary), number of layers, input size and others because the accuracy wasn't getting better when we were improving the model. And the best combinations are:

Optimizer: ‘relu’

Max. validation set acc.: 0.5267

Optimizer: ‘relu’

Max. validation set acc.: 0.5367

In the end, we didn’t reach a better accuracy with LSTM, we were already expecting that because we are working with not a large DataBase and it isn’t a NLP (area that LSTM are more effective).

Recurrent Neural Networks and Gated recurrent units in Neural Networks

Just like in the dense neural networks, when using RNNs and GRU, we selected the 90/10/10 split, and this time started right away with the binary classification problem.

The best score we got using Recurrent Neural Networks was 0.5639 and came from the following combination of parameters:

- Batch size: 128;
- Epochs: 200, with Early Stopping with patience 20;
- Embedding layer, with input dimension equal to 17 and output dimension equal to 32
- SimpleRNN layer with 20 neurons, and activation ‘softplus’;
- Model with optimizer ‘Adagrad’.

And the best score using Gated Recurrent units was 0.5539, resulting from the following combination of parameters:

- Batch size: 128;
- Epochs: 200, with Early Stopping with patience 20;
- Embedding layer, with input dimension equal to 17 and output dimension equal to 32
- GRU layer with 10 neurons, and activation ‘elu’;
- Model with optimizer ‘Adadelta’.

Note: All models and respective accuracies can be found in the zip file ‘models’. In it you can also find a document that relates the model to the number of the plot image.

Conclusion

We began this project very excited with the hopes of managing to build good and accurate predictions that could potentially be used in real life for bets. We began by attempting multi class classification, but got very poor results. By then our hopes had vanished and we began to doubt ourselves and our model. We decided to go for binary classification instead, considering that a draw is equivalent to a lost for the home team. That’s when our hopes came back. This time we had low expectations. The first attempt resulted in lots of research, and even students’ thesis didn’t reach very high scores. In the end, we actually managed to surpass our expectations, and with a very simple model: a dense neural network with just two layers, and twelve hidden neurons. Our maximum validation accuracy was 0.6667, resulting in a prediction accuracy of 0.6102 for the seasons of 2018/2019 and 2019/2020. We tried other approaches such as simple RNN, GRU and LSTM, but unfortunately they resulted in no improvement.

Future work

We believe better accuracies might lay in other Machine Learning algorithms, not necessarily in Deep Learning. But keeping with the Deep Learning approaches, we think gathering more data, especifically on the composition of the teams could help us in improving accuracy. We could also benefit with higher computational power, as we feel it set us back.

We are sure this project doesn’t end here, and we will spend some more time looking at it, trying to improve it and possibly adapting it to other than the Premier League.

References

- Football-data.co.uk. 2020. *Football Betting | Football Results | Free Bets | Betting Odds*. [online] Available at: <https://www.football-data.co.uk/>
- Met Office. 2020. *Historic Station Data*. [online] Available at: <https://www.metoffice.gov.uk/research/climate/maps-and-data/historic-station-data>
- Neil, S., Eaton, V., Eaton, V., Eaton, V. and Johnson, L., 2020. *10 Oldest Soccer Players In The World (Updated 2020) | Oldest.Org*. [online] Oldest.org. Available at: <https://www.oldest.org/sports/soccer-players/#:~:text=Professional%20soccer%20careers%20can%20be,play%20well%20into%20their%2040s.>
- Casas de Apostas Online. 2020. *Placard Apostas Desportivas Com Lucro De 457 Milhões*. [online] Available at: <https://www.casasdeapostasonline.com/placard-apostas-desportivas>
- Develop Paper. 2020. *Prediction Of Football Match Result By Python Machine Learning - Develop Paper*. [online] Available at: <https://developpaper.com/prediction-of-football-match-result-by-python-machine-learning/>
- Vieira, A., Ribeiro, B., 2018. *Introduction to Deep Learning Business Applications for Developers.* [online] Available at: <https://link.springer.com/content/pdf/bbm%3A978-1-4842-3453-2%2F1.pdf>
- Pettersson, D., Nyquist, R., 2017. *Football Match Prediction using Deep Learning*. [online] Available at: <http://publications.lib.chalmers.se/records/fulltext/250411/250411.pdf>
- Neves, T., 2019. *A data mining approach to predict probabilities of football matches.* [online] Available at: <https://repositorio-aberto.up.pt/bitstream/10216/121217/2/343145.pdf>