Aaron Ashery
May 11, 2021

Predicting Soccer Game Outcomes Using Neural Networks

Since my first semester at Sarah Lawrence I have been interested in trying to predict the outcomes of professional sports games. Over two years ago I researched the pythagorean expectation, a formula that outputs a teams expected win percentage based on simple data from that teams previous season. This introduced me to looking at large amounts of sports data, and using it to draw conclusions on the outcomes of whole seasons of sports. The next summer I spent hours every day in a spreadsheet attempting to predict the outcome of the 2019 FIFA Women's World Cup. I inputted all the data I wanted by hand and had it put through my own formulas, which outputted a percentage each team had to win or tie. The point of looking at win percentages rather than just the winner is so it can be compared to sports gambling odds. If the program can accurately predict games with a higher probability than the sports books, the program is a massive success. That summer project did not come anywhere close, but I had a lot of fun messing around with the program and watching to see if my program's predictions came true. After learning multiple coding languages I continued to make similar programs. They ran smoother and took in better data, but none were that accurate still. This semester when I learned what machine learning could do, I knew I wanted to tackle this problem again and see how well new programs could do, with the program itself learning what data is important for predictions rather than what I think is important.

I never expected my programs to reach the accuracy of sports books, however I did hope that they would be much better than my past attempts. My system had two goals. First was to take in mass amounts of data from years of professional soccer matches and try to predict who would win in a match based on the data from the match after it happened. This isn’t very useful for predicting, but I found it very interesting and a good way to start using neural networks. Secondly, the program took data from a season to gather profiles for each team. Then the program was to output a winner and the percentage chance the home team had of winning, when the teams played the next season.

To accomplish these goals I used neural networks. The most time consuming portion of the whole project was sorting a large spreadsheet into smaller csv files with the information I wanted. For the first part of the project the inputs to my neural network were eight numbers. The numbers game from a game played, and each number represented a statistic that took place during the game. The first four were always for the home team, they were shots, shots on target, corners, and fouls. The same four data points made up the rest of the inputs, but for the away team. The system was trained on nearly a decade's worth of games, across eight European countries, each having anywhere from one to three leagues. The targets were what the result of each game was. In the case of what the chances were the home team would win, the targets were the odds given by Bet365, an established sports betting website. For the second part of the project I only looked at data from the 2017-2018 seasons across Europe. With that data each team was given a profile containing 10 data points such as, goals per game, goals against per game, shots per game, etc. For training the network looked to the 2018-2019 season and trained on a little more than the first half of games. For each game there were 20 inputs, 10 from each team which made up each team's profile. Similarly to the first part, the targets were either who won, or the percentage the home team would win.

Overall I ran seven different experiments, which each involved training one network and testing it out. These experiments demonstrated what I explained above. Each experiment had its own journal where I spent a lot of time messing with parameters and using validation sets to find the best results possible for that experiment. In the end I consolidated each experiment down and put them in one notebook. Part one was three experiments. The first was to see how well the network could predict the winner of a game after it happened, but not including games that ended in a draw. The network correctly identified the winner around 77.5% of the time, much better than the 50% rate if it had just guessed. The second experiment was the exact same, but added in draws. The performance of the program was around 57.0%. It dropped from before, but with a guessing rate of 33% now, it still was pretty accurate. Lastly, in part one, when the network was trained on the odds the home team will win, the performance correctly identified the percentage around 30.0% of the time. For this experiment the program outputs a number between 1 and 0, rather than a vector of all 0’s with a single 1 to represent which category it predicts as before. For this

reason I wrote my own function to calculate the accuracy. 30.0% is given by how often the network's predictions were within 5% of the actual percentage. As the tolerance is loosened the program's performance goes up very quickly, at a tolerance of 25% the network has an accuracy of around 90.0%.

For the second and more important portion of the project there were four experiments. The first looked to predict the winner of the last 180 games of the 2018-2019 English Premier League. Using the 20 inputs per match and 200 training games, the network predicted the winner (or if it was a tie) around 50% accurately. Next I opened the data up so that the network tried to predict the 2018-2019 seasons of all the European leagues I had data for. It did this with roughly the same accuracy at 48%. The last two experiments switched back to getting the win percentage for the home team before the match. They also followed the same format as the previous two experiments, first looking at just the English Premier League (EPL) and then all the leagues. With a tolerance of 5%, just looking at the EPL the accuracy was around 45%. With the rest of the leagues it was 35%.

There is a lot to take away from these experiments. I was pleasantly surprised that the performance of the network predicting games before they happened was almost at the same accuracy as the network "predicting" games with data from them. Also, clearly the network learned a lot, since the accuracy improved greatly after it was trained. Another success was that after training the training sets and testing sets were close in accuracy, showing the network was generalized and not overtrained.

Throughout the project, working with the data caused many issues, but by the end product they were all resolved. These issues most of the time had to deal with how the input data was being translated from the main large excel spreadsheet. The obvious thing that "did not work" is that the program does not predict games with enough accuracy to be successful against betting odds. This is largely due to the biggest mistake the network makes which is not being able to classify draws with any good accuracy.

In the future there is a lot to be done to improve the network's performance. One big piece of data that I am sure would help the network would be who the home team is. On average home teams win 50% of the time which leaves the other 50% for two outcomes. It is a big advantage that should have been accounted for. Another improvement needed is to the method of gathering team data used as inputs.

Currently a season's worth of data is used for a teams profile, for the entirety of the next season. As the teams play more games the profiles should be updated, accounting for the new games and reducing the oldest games.

My conference project is in the form of a Jupyter Notebook. To run it you can either go through each cell one by one or just run all cells and look through the outputs. Since there are seven networks, and two of the experiments overlap data, there are 12 files needed to run the notebook (5 data input files and 7 target files).