A Sure Bet: Predicting Outcomes of Football Matches

Sebastien Goddijn
Department of Computer Science
Stanford University
sgoddijn@stanford.edu

Evgeny Moshkovich
Department of Management Science and Engineering
Stanford University
emoshkov@stanford.edu

Rohan Challa
Department of Computer Science
Stanford University
rchalla@stanford.edu

Abstract

Being life-long football supporters we thought it would be interesting to use the deep learning techniques covered in the class to predict the outcomes of premier league games. To complete this task we trained various models including a simple logistic regression, a 3-layer neural network and an LSTM, with the neural network leading to the highest classification accuracy of around 51%.

1 Introduction

Competitive sport is a numbers game, and modern athletic teams are truly beginning to embrace this fact. We are seeing a surge of data driven jobs in the back office of myriad different franchises, and people are beginning to wonder just how effective cutting edge AI and ML techniques can be when applied to an athletic context. Sports have always been a purely human endeavor, and there is an inarguable element of randomness and chance that dictates who the victor will be on any given day, but this begs an interesting question: is there an underlying pattern to this randomness?

For our project we focused specifically on matches in the Premier League, as this is the league that we have watched most closely growing up, and has detailed data that is already publicly available due to the quality of the teams involved.

The input to our algorithm consisted of both a home and away team's individual player ratings, their current record for the season (how many games they had won, drawn and lost), and their win/loss streak. We then used a linear regression, a 3-layer neural network and an LSTM to output a predicted result which could take one of three options: a home win, a draw, or an away win.

2 Related work

A number of people have tried to solve the issue of football match prediction with varying levels of success, and their approaches can be split into two categories.

The first set of papers we read discussed the use of Artificial Neural Networks (ANNs) for sport predictions, with authors using various architectures to achieve state of the art accuracy of just under 60%. One such paper was written by Tax and Joustra [1], who used soccer experts to determine the most effective features for prediction, and through a combination of nine different classification algorithms were able to predict results of Dutch Eredivisie games with a 54% accuracy. We felt that the approach of using ANNs was interesting and had seen a fairly good level of success, but we were curious as to whether or not we could use sequence models to improve the accuracy of our predictions, which led us to papers dealing specifically with Recurrent Neural Networks.

There were a number of interesting approaches in this space, with the use of various different RNN architectures. One paper by Petterson and Nyquist [2], which used LSTM's to predict football match results, was particularly helpful to us in designing our own LSTM architecture. It discussed the use of LSTMs with peepholes, Gated Recurrent Units, and even mentioned the effectiveness of different loss functions for solving this problem, which was extremely useful to us throughout the project.

3 Dataset and Features

Our dataset was an SQLite database that we found on Kaggle [3], which contained information about over 25,000 European Football matches that had happened over the past decade. From this, we extracted all of the data that was relevant to the Premier League, including team, player and match information, and stored these as separate CSV files such that we could manipulate them more easily.

For each match we had access to: the country id, league id, season, stage, date, match api id, home team id, away team id, home team goal, away team goal, home player X positions, away player X positions, home player Y positions, away player Y positions, home player ids, away player ids, and various betting odds (B365H, B365D, B365A, BWH, BWD, BWA. . .).

For each team we had access to: the team id, team name, FIFA id, FIFA date, and FIFA statistics including build up play speed, build up play dribbling, and more. We decided not to use any of these features however, as the data per team was inconsistent and had many missing values.

For each player we had access to: their player id, player name, birthday, height, weight, overall rating, and their FIFA statistics including attacking work rate, defensive work rate, crossing, finishing, and more. We decided only to take their overall rating into account, as the other statistics were highly correlated with this rating.

The final features that we ended up using were the home player X positions, away player X positions, home player Y positions, away player Y positions, home players FIFA overall ratings (11 elements), and the away players FIFA overall ratings (11 elements). We also augmented this with data on the current season, where, for each game, we calculated how many wins, losses and draws the teams had up till that point, as well as their current win or loss streak.

From this we could classify one of three results, where a -1 meant the away team won, a 0 meant there had been a draw, and a 1 meant that the home team had won. Our final data set consisted of 867 home wins, 1390 away wins, and 783 draws for a total of 3040 matches. This spanned across 8 seasons, with 380 games being played in each season. We then decided to use an 80/10/10 split for our training, dev and test sets, for a total of 2432, 304, and 304 games respectively.

4 Methods

4.1 Logistic Regression

For our baseline algorithm we chose to run a basic logistic regression on our dataset. To do this we Xavier initialized a weight matrix of size (3, 22) to avoid issues exploding or vanishing gradient problems, and multiplied it with our input vector which consisted of 22 features (all of the home and away player ratings.) We then normalized this output using a softmax activation function (1), and computed the cost using cross-entropy loss (2) between our predicted output and the real output. We then used gradient descent in order to update our weights such that aforementioned cost was minimized. We ran this for 50000 epochs with a learning rate of $1e^{-6}$.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{n} e^{z_k}}(1) \tag{1}$$

$$L(\hat{y}^{(i)}, y^{(i)}) = -\sum_{j=1}^{n}(y_j^{(i)} log(\hat{y}_j^{(i)})) \tag{2}$$

4.2 Neural Network

The second model we experimented with was a fully connected three-layer neural network. Our first hidden layer had 27 nodes, our second had 9 nodes, and our third hard 3 nodes, with all of the weights for these layers being Xavier initialized once again to avoid exploding or vanishing gradients. The neural network takes our input vector, and, at each layer, multiplies it with the associated weight matrix for that given layer. Once it has done this multiplication, it runs it through a ReLU activation (3) before passing it on to the next layer. After passing through the last layer, we use a softmax activation (1) to normalize the output, and then use cross-entropy loss (2) to calculate the cost of that iteration. We then used Adam to update our weights to minimize the aforementioned cost. We ran this for 30,000 epochs with a learning rate of $1e^{-6}$.

$$R(z) = \max(0, z) \tag{3}$$

4.3 Long Short-Term Memory

A Long Short-Term Memory (LSTM) unit is a recurrent network unit that is designed to remember values for either a long or a short duration of time. For example, if the LSTM unit detects an important feature from an early input sequence, it carries this information over a long distance. This is significant for many applications, such as speech processing, music composition and time series prediction. Since a LSTM does not use an activation function within its recurrent components the stored value is not iteratively squashed over time. To be able to use this LSTM we needed to sequence our data, and as such chose a sequence length of 38, as each team in the league played 38 games in a season, which gave us 160 samples to work with. From here we passed our input features to a hidden LSTM layer with 32 nodes, and, after using dropout with a keep-probability of 0.5, we passed the output from this layer to our softmax activation function (1). We once again used a cross-entropy loss (2) to calculate the cost for each iteration, and used Adam to update our weights and minimize this cost. We ran this with a batch size of 10 (to speed up training) for 300 epochs.

5 Experiments/Results/Discussion

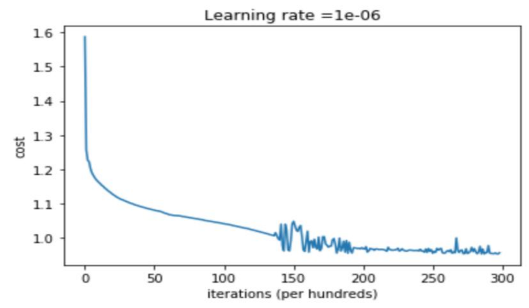
5.1 Logistic Regression

As mentioned previously, we ran our logistic regression on an input of 22 features, containing the FIFA ratings of all 11 of the home and away players for each match. We chose the learning rate of $1e^{-6}$ and 50,000 epochs based on a few simulations that we ran, as these led to the best results. This led to a training accuracy of around 35% and a dev accuracy of 34%, which is slightly better than random given our three output classes. It did manage a good spread of results, as can be seen in the confusion matrix below, but clearly there was room for improvement.

	Home win	**Draw**	**Away Win**
Home Win	16	26	36
Draw	32	22	33
Away Win	30	44	65

5.2 Neural Network

To find the optimal neural network we tested a number of alternative architectures, though we kept the depth of the network constant. We considered two different learning rates, $1e^{-6}$ and $1e^{-5}$. We considered three different options for the number of nodes in each layer, (20, 10, 3), (12, 6, 3) and (27, 9, 3), and we ran all of these networks for 20, 30, 40, 50, 60, and 100 thousand iterations to determine which gave us the best result. We compared them based on their accuracy on the development set, and our best performer had a learning rate of $1e^{-6}$, a node structure of (27, 9, 3), and was run for thirty thousand iterations. This initial implementation gave us a training accuracy of 51% and a development accuracy of 48%, with the loss over time graphed below.



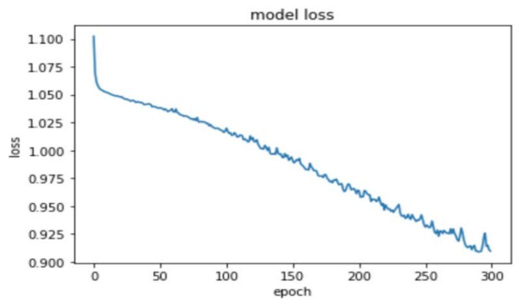
Once we reached this level of accuracy, we were curious to see if incorporating the 'momentum' of a given team would have any impact on the accuracy of our predictions. As such, we calculated the number of wins, draws and losses a team had at the point at which they played a given game, as well as their current win or loss streak. We then used the best performing network architecture determined above, and achieved a training accuracy of 55% with a development accuracy of 51%. However, this increased accuracy meant that our model no longer predicted any home wins, as can be seen by the comparison of the confusion matrices for these two models below. Despite this interesting development, we believed that this demonstrated that past data did have an effect on the accuracy of our prediction, which prompted us to explore how effective an LSTM could be in solving this problem, as it takes into account previous information by default when making its prediction.

	Home win	**Draw**	**Away Win**
Home Win	22	30	26
Draw	26	45	16
Away Win	28	32	79

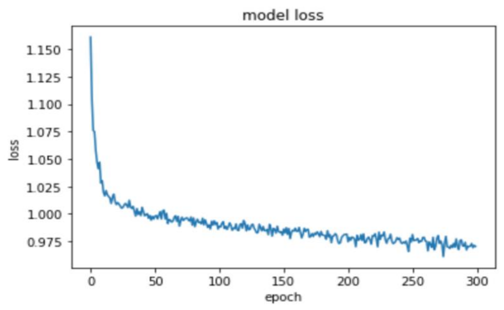
	Home win	**Draw**	**Away Win**
Home Win	0	20	58
Draw	0	39	48
Away Win	0	22	117

5.3 Long Short-Term Memory

To find the optimal architecture for our sequence model, we once again tested a number of possibilities, and used their development accuracies to compare them to one another. We tested architectures with both 1 and 2 LSTM layers, which had either 16, 32, 64, 256, or 512 nodes per layer. We ran these architectures for 100, 150, 300, and 500 epochs respectively, and eventually chose a one-layer network with 32 nodes in that layer, trained for 300 epochs. This gave us a training accuracy of 56.81% and a development accuracy of 37.17%. From these accuracies, and the graph below, we could clearly see that the model was over fitting the training set. When we ran the model for 500 iterations for example, it wasn't uncommon to see a training accuracy of over 90%!



As such, we knew we needed to adopt some regularization techniques to reduce this issue of over-fitting. For our new architecture we limited our search to some of the best performing architectures we had seen in our initial LSTM model, for example we had seen that a 1-layer LSTM outperformed a 2-layer model in almost all cases so we did not re-explore these architectures. This time, however, we incorporated a dropout with a keep probability of either 0.25, 0.5, or 0.75 after our LSTM layer, and eventually settled on a best performing network with 64 nodes in the LSTM layer and a dropout of 0.5. This gave us a training accuracy of 54.24% and a development accuracy of 47.34%, which was much better than our original model. We could also see from our graph displaying the loss over time that this network was suffering less from the issue of over fitting, as it was a much less linear downward slope.



6 Conclusion/Future Work

Model	Train Accuracy	Dev Accuracy
Logistic Regression	0.3573191	0.33881578
Neural Network	0.54893094	0.5131579
LSTM	0.5424	0.4737

To conclude, our three-layer neural network was the most successful algorithm that we tested. Despite the fact that historical data seemed to improve the accuracy of our prediction, the LSTM was unable to improve on the neural network. If we had more time we would try and gather more data on which to train, in order to reduce our algorithms' tendency to overfit the training data. Though we had some success reducing this using dropout, an increased amount of data would have been even more beneficial. Additionally, we would like to experiment with different architectures and different LSTM variations, to see if these would improve on our classification accuracy, and maybe allow us to predict not only the result of various games, but also the scoreline of these matches.

7 Contributions

We felt that the work was evenly distributed between the team and everyone pulled their weight. Evgeny was primarily responsible for doing background research on the various implementations people had already worked with in trying to solve this problem, as well as being the driving force behind the work we did on the poster. Rohan and Seb evenly split the coding work to actually get the model up and running, and all three of us contributed to this final write-up evenly. We really enjoyed working together as a team and are proud of what we achieved during this course.

References

[1] Tax, Niek, & Yme Joustra. "Predicting the Dutch football competition using public data: A machine learning approach." *Transactions on Knowledge and Data Engineering 10.10* (2015): 1-13.

[2] Pettersson, Daniel, and Robert Nyquist. "Football Match Prediction Using Deep Learning." (2017).

[3] https://www.kaggle.com/hugomathien/soccer

[4] Bunker, Rory P. & Fadi Thabtah. "A machine learning framework for sport result prediction." Applied Computing and Informatics (2017).

[5] McCabe, Alan, & Jarrod Trevathan. "Artificial intelligence in sports prediction." Information Technology: New Generations, 2008. ITNG 2008. Fifth International Conference on. IEEE, 2008.

[6] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.