PIC 16B Course Project Report

By Jude Baguio, Ishaan Garg, and Zackary Rodriguez

Introduction

Our project aims to analyze data from individual soccer matches in three English Premier League seasons: 2017-2018, 2018-2019, and 2019-2020, which will be cleaned, collated and placed into a machine learning model with TensorFlow that predicts the amount of goals a team would score. These match statistics will include the following data points: possession, passing accuracy, shots on target, goalkeeper saves, tackles, fouls, interceptions, goal kicks, corners, and crosses. We then restructure our data for one season (2019-2020), in the form of a directional neural network, and use PageRank scores to recreate a league table, which we compare with the actual league table. Our analysis wishes to show which data point best predicts goals scored, and whether goals scored can be an effective indicator of league position.

Files

The files used for the project are the following:

- project.pdf for report
- football_scrapy folder, contains football_spider.py
- Data Cleaning.ipynb for data cleaning
- modelinterp.ipynb for model creation and interpretation
- predmodel.ipynb for final goals predictor model
- Prem_Page_Rank.ipynb for PageRank interpretation of dataset

Other files included in the folder are files generated within these program files, and can be obtained by running the above files.

Data Collection with Scrapy *(Zack)*

We collected the data that would be used to train our model from fbref.com, an online database that records statistical information relating to soccer leagues across the world. We generated a scrapy project, which can be found in our submission as a folder named 'football_scrapy', and within this project we wrote a spider, 'football_spider.py', which was originally designed to scrape information off of various match report pages, all of which were linked to the starting url,

'https://fbref.com/en/comps/9/3232/schedule/2019-2020-Premier-League-Scores-and-Fixtures'.

The spider class we wrote has three methods, start_requests, parse, and parse2. The first method simply initializes the scraping process by yielding a scrapy.Request object with a callback function of self.parse for each url in a url list, which when the spider was first written, was only the above url.

The first parse method collects all of the url paths to each match report page, which are available as href information in a table at each of the starting urls, as a list using response objects and css selectors. Once the urls are collected, for each url path collected, this parse method creates a full_url using urljoin, and then creates a request object for the match report page, with parse2 as the callback function.

The parse2 function does most of the heavy lifting in terms of collecting match data. The match report web pages, from which parse2 collects data, have information organized into two tables. One larger table stores team names and major statistical categories along with graphics that correspond to different statistical distributions, for example a bar that is filled in with the percentage of possession one team had in the match. Beneath this, is a second table that stores more minor statistical data purely textually. In both of these tables, statistical information is split into two columns, home team and away team, as such, each data point in the resulting file has two columns for each stat, home and away. The following are all of the statistical categories that our spider collects: team name, goals, xG (expected goals), possession, passing accuracy, shots on target, shots, saves, fouls, corners, crosses, touches, tackles, interceptions, aerials won, clearances, offsides, goal kicks, throw ins, long balls.

For the most part, data collected from the major stats table required one response.css() function for each statistical category, with a few exceptions such as shots on goal and shots being collected together, as well as possession and passing accuracy. All of the information in the lower table was able to be collected as a list with a single response.css() object, although this object also collected some unnecessary text information, so not every element of this list is stored. The final output of parse2 is a dictionary that stores each statistical category for each team in a given match.

After the spider was written, we used it to collect the statistical data for the 2019-2020 season in a file called 'football.csv'. Later in the course of our project, we decided to try using a larger sample size to train our model. To do this, we wrote two more urls into our starting list, which correspond to the 2017-18 and 2018-19 premier leagues seasons. Because these webpages

had the same structure as the original starting url, we were quickly able to collect this data into ‘prem.csv’, another file with the same structure as ‘football.csv’ but two more seasons of data.

Data Cleaning and Preprocessing with Pandas *(Ishaan)*

Data cleaning and preprocessing was entirely carried out in a Jupyter notebook file titled ‘Data Cleaning.ipynb’, using the Pandas module.

After reading the scraped data stored in ‘prem.csv’ into a Pandas dataframe, there were 1140 rows and 40 columns, where each row corresponds to a premier league match. The first 20 columns were attributes/statistics of the home team while the latter 20 columns were attributes of the away team, for each match. Since the aim of our project is to build a machine learning model that estimates the number of goals each team scores in every match (which is then compared to the ground truth values in the expected goal (xG) column), which team was the ‘home’ team and which team was the ‘away’ team is irrelevant. So, we append all the ‘away’ data to the ‘home’ data, and change the relevant column titles accordingly. The result is a reshaped Pandas data frame with 2280 rows and 20 columns (half the columns and double the rows).

Next, we notice that the data type of the ‘possession’ and ‘passing_accuracy’ columns are strings, so we convert the type of these columns to float64 by ‘stripping’ the percentage symbol (%) and subsequently dividing the number by 100 so that it is still expressed in percentage terms. This is done so that the machine learning model is able to properly interpret these statistics as percentages (floating point values) rather than meaningless strings.

Finally, we convert the resulting Pandas dataframe into a .csv file titled ‘stats.csv’ so that it may be inputted into a machine learning model.

Model Creation with TensorFlow *(Jude)*

After the gathering and cleaning of data for 3 seasons, we pursued a machine learning model to find a relationship between goals scored and the variety of statistics from FBRef. We utilized a TensorFlow Keras Sequential model consisting of multiple dense layers after preprocessing the variables. The following processes can be found in the Jupyter notebook 'modelinterp.ipynb'.

The first action was to optimize our inputs for the Keras model. To proceed, we first analyzed the Pandas dataframe. We converted the float-valued columns "xG", "possession", and "passing_accuracy" to float32 to obtain the performance benefits of 32-bit floating notation (instead of the larger-size 64-bit floating notation). We inspected the data to ensure that each team played against each other 38 times in a single season.

The preprocessing continued with the splitting of the input data into training, test, and validation data. This was ideal for testing our model later on. We then normalized the training, test, and validation features using scikit-learn's preprocessing.StandardScaler(), through the fit_transform() and transform() methods. Data normalization ensures that we standardize anomalies that could make analysis of the data more complicated.

To create the Sequential model itself, numerous layers were trialled. We tried various numbers of layers that utilized different activation functions, with differing units for each layer. The most effective activation functions turned out to be "selu" and "elu", which we incorporated into our model. We realized that having 4 dense layers was optimal for the model, and so incorporated this to the model as well. Although we attempted to include dropout layers and BatchNormalization() techniques, these did not improve our model, and in fact, increased the model's loss value. As such, we omitted these techniques from our model. The final model was analyzed and selected based on the loss value returned by evaluate(), as we attempted to minimize this parameter.

We had two iterations of our model. The first model utilized 6 features that were deemed most important to goals scored - "possession", "passing_accuracy", "shots_on_target", "saves", "tackles", and "fouls". After analyzing and interpreting the model (discussed in the next section), we found that the entire model inadequately explained how a change in any of these factors affected the amount of goals scored. We then incorporated all 10 features by adding "interceptions", "goal_kicks", "corners", and "crosses", to a second model. As the number of inputs differed for each model, the units utilized within the dense layers varied slightly. The particular details of each model are as follows:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 6)                 42
_________________________________________________________________
dense_1 (Dense)              (None, 30)                210
_________________________________________________________________
dense_2 (Dense)              (None, 30)                930
_________________________________________________________________
dense_3 (Dense)              (None, 12)                372
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 13
=================================================================
Total params: 1,567
Trainable params: 1,567
Non-trainable params: 0
_________________________________________________________________
```

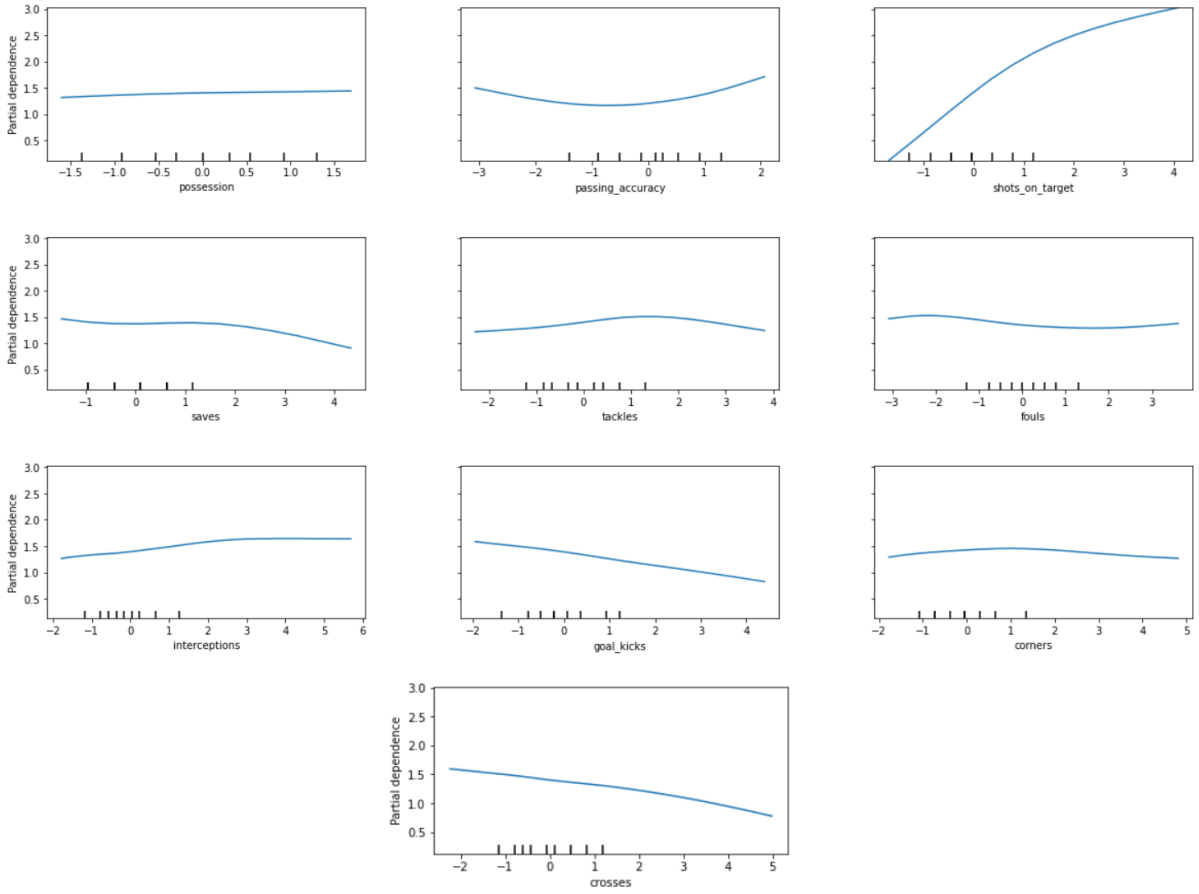
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 10)                110
_________________________________________________________________
dense_6 (Dense)              (None, 60)                660
_________________________________________________________________
dense_7 (Dense)              (None, 30)                1830
_________________________________________________________________
dense_8 (Dense)              (None, 12)                372
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 13
=================================================================
Total params: 2,985
Trainable params: 2,985
Non-trainable params: 0
_________________________________________________________________
```

We use the second model moving forward.

Model Interpretation with Scikit-learn *(Jude)*

To interpret our model, we utilize the Scikit-learn package, inspection. We use this to create a partial dependence plot of each variable, relative to the predicted outcome of goals scored. The following processes can be found in the Jupyter notebook 'modelinterp.ipynb'.

This partial dependence plot shows us the marginal effect each feature has on the number of goals scored. The PDP plots for the 10 features are as follows:



The PDP plots all show a relatively modest relationship between each variable and goals scored, except for shots on target. The higher the shots on target taken by a team, the larger the predicted number of goals scored. This makes intuitive sense, as more shots on target gives a team a higher chance of scoring goals. In fact, if shots on target were 3 standard deviations higher than average shots on target, teams would be predicted to have nearly 2.5 goals - which guarantee a win in most games.

Some variables exhibit a modest effect on predicted goals scored. There is a modest uptick in passing accuracy, where an accuracy 1 standard deviation higher than average leads to a prediction of 2 goals. Having a larger passing accuracy implies more control of the game, which tends to lead to more goals. However, the increase cannot be huge, as there are teams in the league who pass the ball in frustration as the opposing defence sits back to defend resolutely.

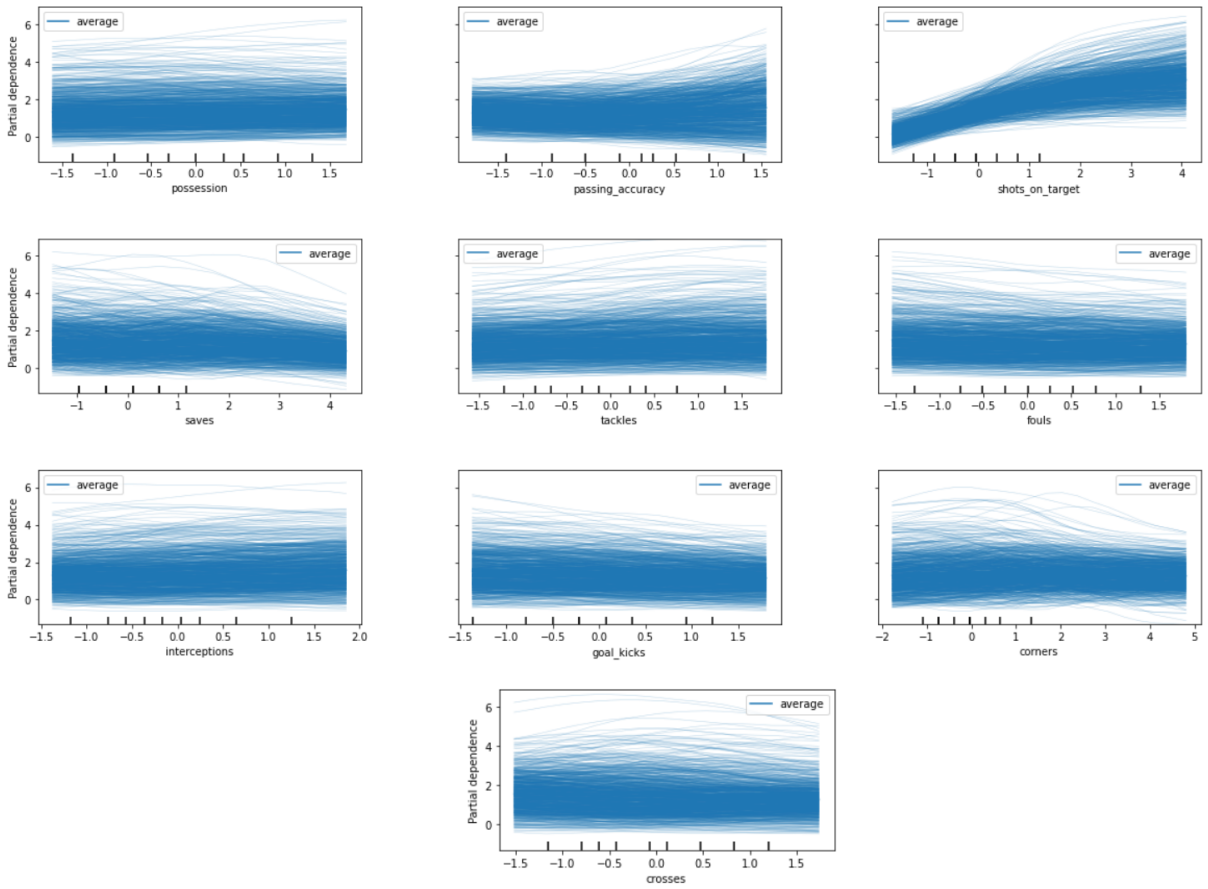
This prevents passing accuracy from completely implying a higher amount of goals scored. Furthermore, we see that an accuracy 1 standard deviation lower than average leads to somewhat higher goals scored - this is representative of defensive teams who instead of passing heavily, play their game by defending resolutely.

What is surprising are the variables which show a decrease in the prediction. In particular, saves and goal kicks decrease the predicted goals scored. Although making a large number of saves is a hallmark of good defence and goalkeeping, it also implies that a team is on the back foot - that they face pressure from their opponents, and are less likely to have control of the football to create chances and score goals. This logic is applicable to goal kicks - they are only made when opponents kick the ball out of play on the defence's end of the pitch. As such, the PDP plot's interpretation is valid.

Corners and crosses are made when a team attacks, and so would usually be associated with goals scored. Yet PDP plots show that predicted goals scored actually decrease with larger numbers of corners and crosses. This is a very surprising statistic - many teams in the Premier League, such as Burnley, play their offence through crosses, and many successful teams, such as Liverpool and Manchester City, play by taking many corners. This implication could stem from the way football is generally played - by thoroughly passing within a team until an opening is found through an opponent's defence. This primary style of play is counterintuitive to taking many corners and crosses, leading to the negative relationship seen between these variables and goals scored.

One intriguing variable is tackles - having an average number of tackles leads to the worst prediction outcome of goals scored. This highlights the dual nature of teams' style of play. Teams that have a good control of the game, and hold the ball for large spells, do not need to make tackles, and such teams tend to score. On the other hand, teams that are known for sitting back on their end of the pitch, and defending at all costs, make large numbers of tackles, but are nevertheless good teams that have the propensity to score. It is teams who do not lay on either end of the spectrum that are not considered to score as much, which shows in the variable's PDP plot.

There are three variables with PDP plots that show minimal change to predicted goals scored - possession, fouls, and interceptions. To further analyze these variables, we use Individual Conditional Expectation (ICE) plots, which display one line per sample for each variable that shows how the instance's prediction changes when a variable changes. These ICE plots could reveal heterogeneous relationships that are masked in the averaging process of PDP plots. The ICE plots for the 10 features are as follows:



All ICE plot curves of the variables discussed so far seem to obey the relationships laid out by the PDP plots. For possession, there seems to be very little marginal change to the prediction for most samples. As such, we can conclude that possession is not an important feature in predicting goals scored. This holds true in real-life analyses of football matches, as the best teams are not necessarily those that have the highest possession rates. There are great teams that play defensively and score on the counter.

For fouls, there are samples where predicted goals increase with fouls, and others where predicted goals decrease. This is interesting because we expect fouls to indicate frustration with the current game - and so would not correspond with more goals. On the other hand, some teams may have players known for being brash in the field - who would make fouls regardless of whether they are doing well or not. This accounts for most of the samples, where no discernible marginal change is found. For samples where fouls increase predictions of goals scored, more analysis would be needed. However, some teams begin to play more defensively when ahead in goals as part of their tactics, which implies a more brash style of play and more fouls, which may explain these particular samples.

For interceptions, most samples show a modest increase in their predictions, yet there are a few which show a decrease in predictions. Overall, these samples balance out to the relatively flat PDP found for the variable. For most samples, a decrease in predictions make sense, as

interceptions are usually performed by teams who do not have the ball, and so are defending from their opponents at a consistent rate. However, once again appealing to the unique defensive styles of play certain teams employ on the field, there are defensive-minded teams who score many goals and rely on defensive tactics, such as interceptions, for their gameplay. These teams comprise the few samples where interceptions increase predictions.

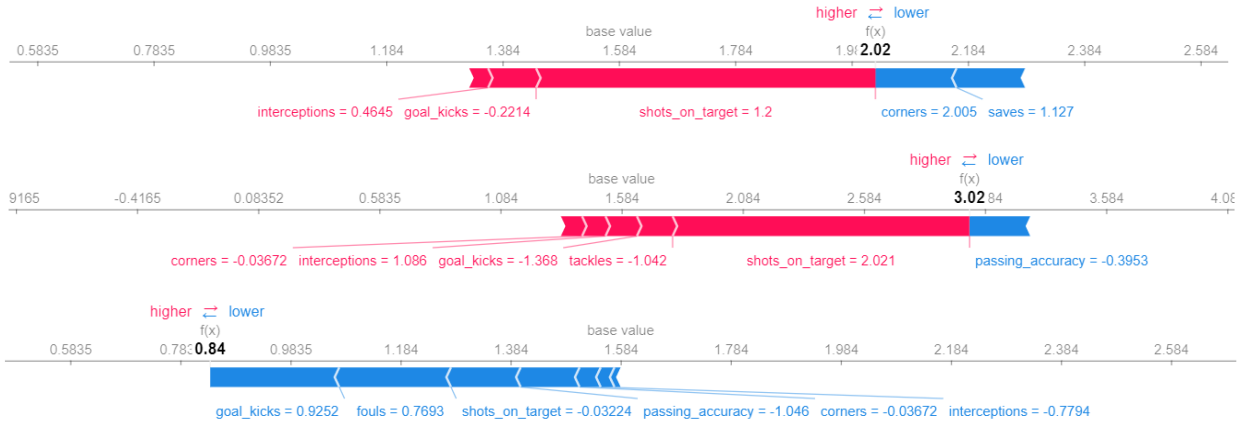
Our analysis of PDP and ICE plots reveal that shots on target was the most potent explanatory feature, which was expected. Although the rest of the features did not comparatively show as strong a relationship as shots on target to goals scored, they highlighted the myriad of defensive and offensive tactics employed by teams to score goals, and therefore win games. Each feature contributed to the number of goals scored, which was seen when individually analyzing each sample.

Model Interpretation with SHAP *(Jude)*

To better understand how much each feature contributes to our prediction of goals scored, we implemented SHAP to our model. The package we used, KernelSHAP, estimates for every sample instance the contributions of each feature value to the prediction. This allows us to discern which features contributed significantly to our prediction. The following processes can be found in the Jupyter notebook 'modelinterp.ipynb'.

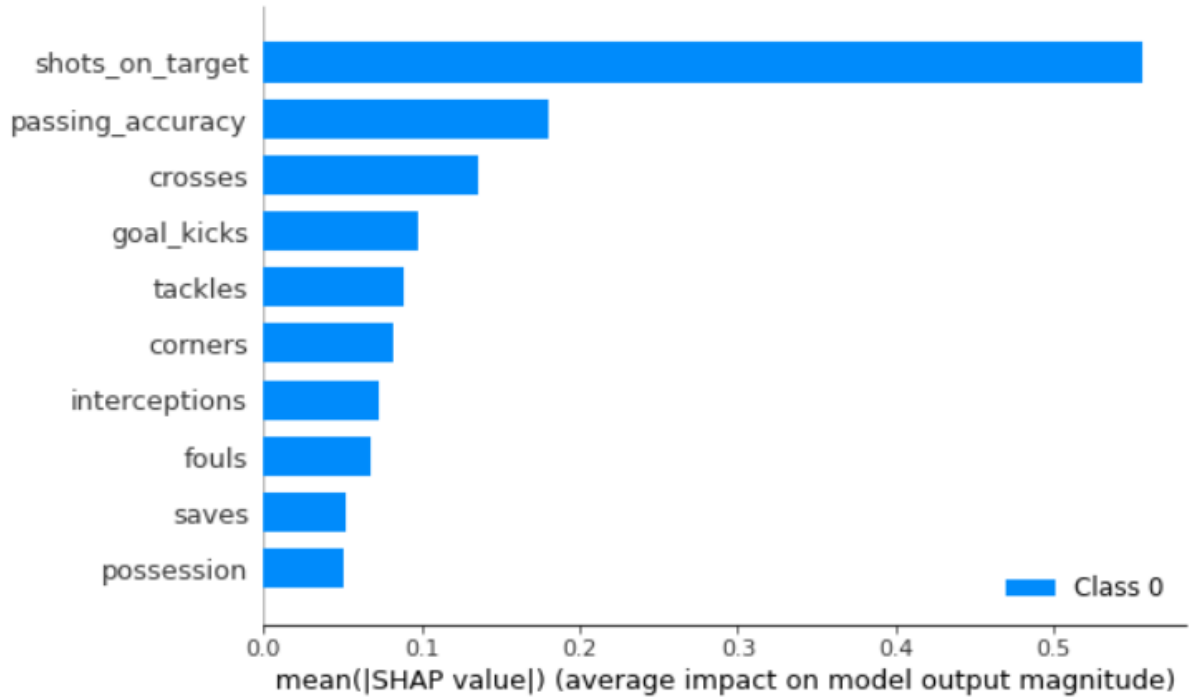
We have interpreted SHAP values for individual values (to observe how each feature contributes to the prediction of one sample), and as a collective (to observe the overall strength of a feature in model predictions). As SHAP values take a long time to calculate, we restrict our dataset to 100 samples for the KernelExplainer.

Each sample has its own force plot. We show an example of some samples and their SHAP values using the three individual random samples below



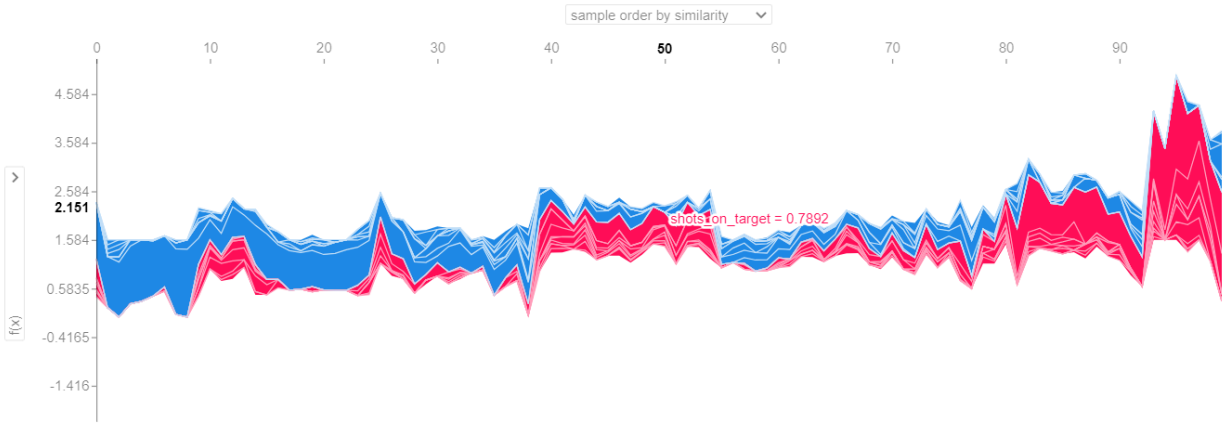
It seems that shots on target always has a major contribution to the prediction of samples, whether it be a positive or negative contribution. This makes sense, as we identified shots on target as the feature with the most obvious relationship to goals scored in our PDP analysis. There are other features, such as passing accuracy and goal kicks, which contribute moderately in the three samples we show. However, these features' contributions are not omnipresent in all samples. As such, we can deduce that shots on target has a large contribution to the prediction, but we are inconclusive of the contributions of other features.

To look at feature contribution more holistically, we use a summary plot to show each feature's SHAP Value distribution. The summary plot is shown below:

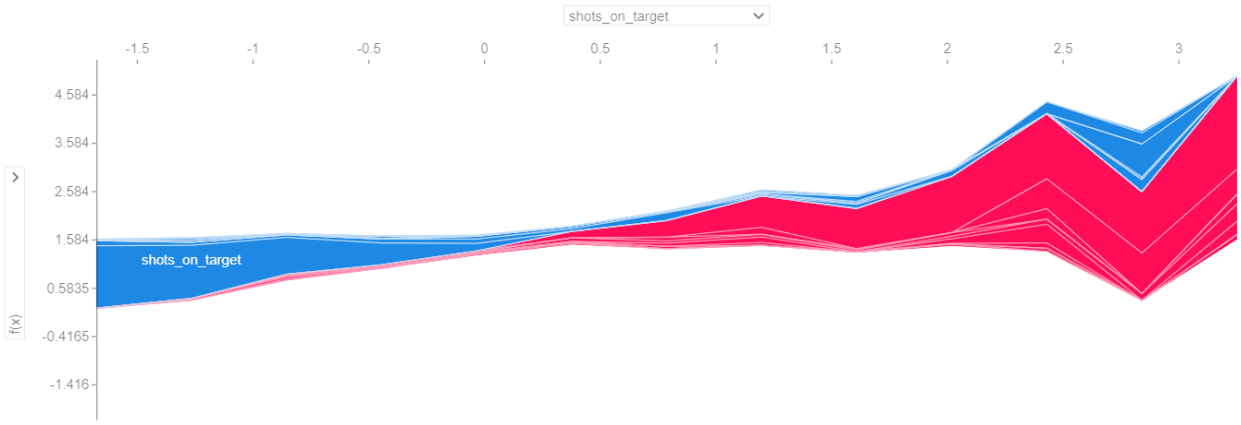


As evident, shots on target is confirmed to be the most potent feature, followed by passing accuracy and crosses. These results are similar to our analysis using PDPs and ICE plots.

To solidify our analysis of each feature's contribution, we can form a collective force plot. A plot that aggregates individual force plots is then created.



The above is an interactive collective force plot found in "modelinterp.ipynb". The x-axis and y-axis can be altered to look for individual effects of features on the prediction. An example is provided with the shots on target feature.



In the collective force plot with shots on target as the x-axis, we see that the feature contributes to a higher prediction of goals scored as shots on target increases. This contribution is pronounced when shots on target are around 2.5 standard deviations higher than the mean. On the other hand, shots on target contributes to a lower prediction of goals scored when they are 1 standard deviation lower than the mean. This is also expected - lower shots on target should correspond to lower number of goals.

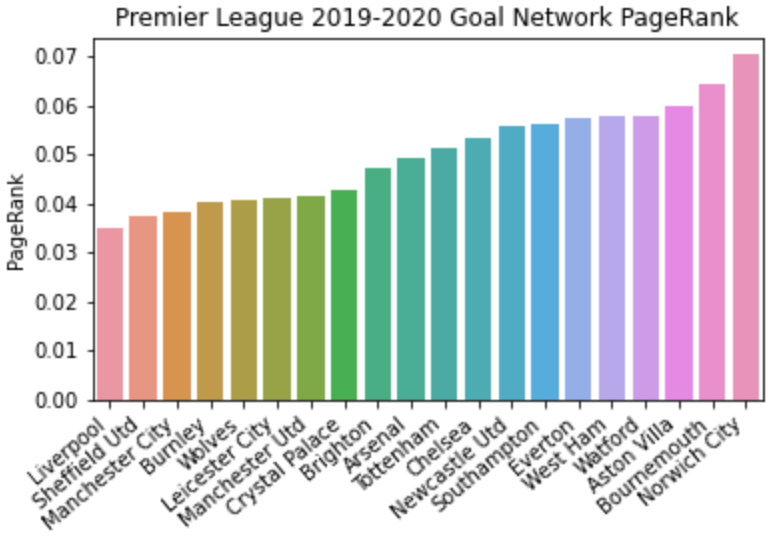
Our analysis of the SHAP values demonstrates the importance of shots on target as a predictor of goals scored. We also found that high passing accuracy seems to be a key feature of high-scoring teams, which suggests it may be more reliable as a predictor than we originally thought. All in all, our findings confirm our analysis of the PDPs and ICE plots.

Data Interpretation with PageRank *(Zack)*

When constructing our machine learning model, we considered goals as the output of a regression function which took as input some number of statistics. In order to contextualize the results of our model, we sought to examine the directional context of a goal being something that one team scores on another. In order to do this, we had to restructure our data into the form of a directional neural network. The assembly of this network, as well as the graphical representations of our data that we were able to construct from this network can be found in the file 'Prem_Page_Rank.ipynb'.

The network was built from information in 'football.csv', the table which contained the statistical information that was scraped from the 2019-2020 Premier League season only. The decision to use only one season was made because English Football's relegation and promotion system means that the same 20 teams were not present in all three years of data that we collected. This data was loaded as a dataframe in Pandas, and subsequently converted into a table with columns 'score_team', 'target', and 'goals'. From this dataset, it was possible to use nx.from_pandas_edgelist() to construct a directional graph displaying the goals each team scored on each other team during this season, with number of goals acting as a weight parameter which determines the thickness of the edges:

Because of the quantity of goals that were scored and the variety of teams, it is difficult to draw any meaningful conclusions from this representation. However, we were able to use the network this graph represents to find the PageRank information for each of the teams in this season. The following PageRank data was generated using the NetworkX PageRank function, with an alpha value of 1 representing the fact that this network has no teleportation, as when two teams are playing, one of those teams can't score on a random third team.



We were also able to draw conclusions about this season by comparing our PageRank scores for each team, from ordered from least to greatest, with the 2019-2020 Premier League table:

P	TEAM	P	W	D	L	GF	GA	GD	PTS
1	Liverpool	38	32	3	3	85	33	52	99
2	Manchester City	38	26	3	9	102	35	67	81
3	Manchester United	38	18	12	8	66	36	30	66
4	Chelsea	38	20	6	12	69	54	15	66
5	Leicester City	38	18	8	12	67	41	26	62
6	Tottenham Hotspur	38	16	11	11	61	47	14	59
7	Wolverhampton Wanderers	38	15	14	9	51	40	11	59
8	Arsenal	38	14	14	10	56	48	8	56
9	Sheffield United	38	14	12	12	39	39	0	54
10	Burnley	38	15	9	14	43	50	-7	54
11	Southampton	38	15	7	16	51	60	-9	52
12	Everton	38	13	10	15	44	56	-12	49
13	Newcastle United	38	11	11	16	38	58	-20	44
14	Crystal Palace	38	11	10	17	31	50	-19	43
15	Brighton and Hove Albion	38	9	14	15	39	54	-15	41
16	West Ham United	38	10	9	19	49	62	-13	39
17	Aston Villa	38	9	8	21	41	67	-26	35
18	Bournemouth	38	9	7	22	40	65	-25	34
19	Watford	38	8	10	20	36	64	-28	34
20	Norwich City	38	5	6	27	26	75	-49	21

	PageRank
Liverpool	0.034913
Sheffield Utd	0.037483
Manchester City	0.038369
Burnley	0.040490
Wolves	0.040863
Leicester City	0.041285
Manchester Utd	0.041591
Crystal Palace	0.042864
Brighton	0.047207
Arsenal	0.049382
Tottenham	0.051257
Chelsea	0.053623
Newcastle Utd	0.055833
Southampton	0.056350
Everton	0.057649
West Ham	0.057855
Watford	0.057990
Aston Villa	0.059952
Bournemouth	0.064526
Norwich City	0.070515

It is worth discussing how this PageRank data was configured in order to draw meaningful conclusions from the varying PageRank scores of these teams. Because goals scored are directional edges, teams that allow fewer goals would be expected to have a lower PageRank. This is consistent with our findings, as Sheffield United, one of the best defensive teams in this season, has the second lowest PageRank in this set (0.037483). In addition, a high goal scoring team would expect to have a depressed PageRank score, as the function would interpret this as many possible avenues away from a node. This pattern is again confirmed by the lowest PageRank team, Liverpool (0.034913), which secured the Premier League championship with it's potent goal scoring offense in this season.

Overall, our PageRank data gives us a clear picture of how goal-scoring affected a teams overall performance in this season. The highest PageRank team, Norwich City, was relegated from the premiership in this year, indicating that the distribution of goals within a season is one of the most important factors for determining the outcome of a season of football. There are a few teams like Chelsea, that seemed to perform better in the league table than their goalscoring network position would indicate, as well as teams like Burnley, who were able to secure a position towards the top of the goalscoring network while underperforming in the league. This variation is likely due to our network weighing differently depending on the PageRank of the team the goal is scored against. In addition, the Premier League weights wins and losses the same regardless of goal margin, which only comes into play in the case of a points tie. Despite this, many of the trends that are visible in the prem table can be seen through our pagerank data, as mentioned above. When considering this information in combination with our model, which examines the factors that go into scoring goals within a game, we are able to gain insight into the last three years of English Premier League football.

Conclusion

The original idea behind this project was to create a model that could predict how many goals a football team would score, given a set of match stats. The final iteration of our project includes that model, but it also includes the conclusions about goal scoring we were able to draw by interacting with our model. Both of our model interpretation processes identified shots on goal as the single statistical category that is most important in determining the number of goals scored. In addition to the powerful effect of shots on goal, a number of other interesting trends can be found when examining how soccer statistics contribute to goals. Patterns such as both low and high possession games both causing an increase in goals illustrate how competing football philosophies can both have success.

The nature of football, a sport where long stretches of great play are meaningless if one player misses the resulting shot, makes a regression model like the one we built an imperfect way of examining goals. However, the fact that the findings of our model can be contextualized in terms of conventional football knowledge and tactics lends a measure of credibility to our project in terms of being meaningful football analysis. Our further analysis with PageRank helps place our conclusions about goal scoring into the larger picture of a Premier League season. Perhaps the outcome of this analysis, that the way teams score goals affects their league performance, is not exactly groundbreaking, but the differences between the PageRank data and the table help demonstrate how team performance and result can become disconnected at both the game and the season level of English football. In the end, this exploration of the uncertainty that can exist within an entirely human-made system like football has helped deepen our understanding of the advantages and limitations of data analysis with Python.