Capstone Project

Predicting the Outcome of Premier League Matches

Avi Mukesh

March 2023

1 Project Definition

1.1 Project Overview

The Premier League is the most watched sports league in the world. 20 teams battle it out by playing two matches against every other team - one home and one away. At the end of each season teams are relegated from the league if they finish in the bottom 3. Teams are also promoted from the Championship.

My goal is to look at Premier League data over the past decade and train a model that predicts which team will win in a given fixture. The input to the model will be the ids of the two teams, along with other information such as current position in the table, and the number of clean sheets kept in their last 5 games. The model will output either a win, draw, or a loss for the home team as the outcome of the match.

I'm using a dataset from Kaggle that has been scraped from the Transfermarkt website. The dataset is called "Football Data from Transfermarkt" [1].

1.2 Metrics

I will use two metrics to evaluate the model: accuracy and precision. The problem is essentially a 3 class classification problem, so I can't simply use the precision_score. I will use "weighted" averaging for calculating precision. I used a weighted average because there are are much fewer draws compared to wins, but I would like all classes to contribute to the metric fairly. I use precision as a metric because in this scenario, the cost of a false positive is high. Out of all the wins that are predicted, it is important that a large proportion of these are actual wins. Similar for losses and draws respectively.

2 Analysis

2.1 Data Exploration

The data contains this list of files

- appearances.csv
- club_games.csv
- clubs.csv
- competitions.csv

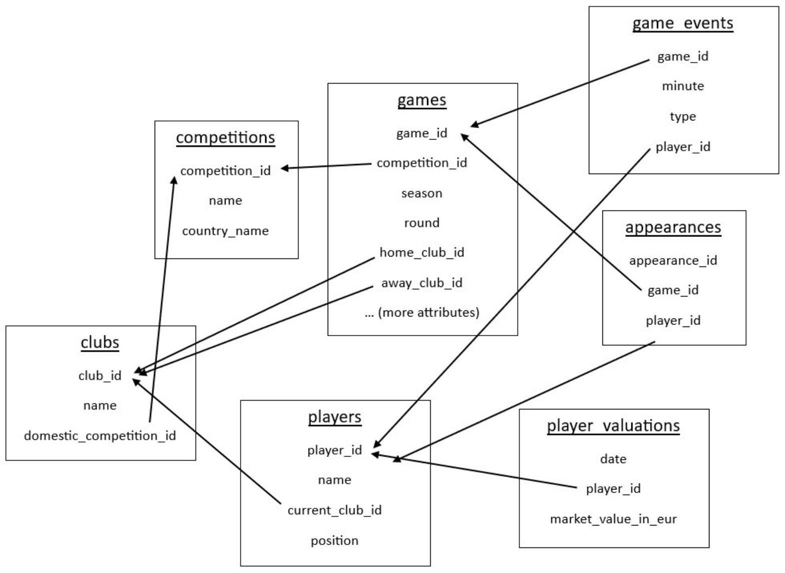

Figure 1: Entity relationship diagram

- game_events.csv
- games.csv
- player_valuations.csv
- player.csv

One key thing to keep in mind is that the club position attribute in games is the position of each side after the match ended. Nonetheless, this column will be close to the team's position before the start of the match, so can be used as a reliable predictor for our use case.

See figure 1 to see how all the tables link together. Note that not all attributes are included, only the ones that are important to us.

2.2 Data Visualisations

I was first interested at which points in a football match are goals mostly scored. For this I looked at the game_events file which contains two types of game events: goals and substitutions, and the game minute at which they occur. Figure 2 shows that there is a spike in the number of goals scored just before and after half time and almost in the closing minutes of the second half.

I investigated whether there is any link between the referee and the number of goals that are scored. On average, the number of total goals scored in a match is 2.8. All referees lie close to this number, so there likely isn't much reason to suspect the choice of referee affects the number of goals scored in a game.

Finally, I wanted to see whether the number of fans in the stadium had any impact on the number of goals scored. This is particularly relevant as games were played behind closed doors in the 2020/21 season and for parts of the season before. This resulted in an influx of goals being scored in games [2].

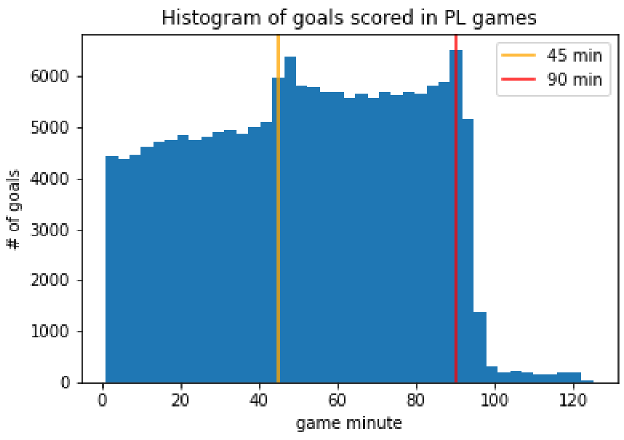

Figure 2: Histogram showing distribution of 16,000 goals scored in the PL

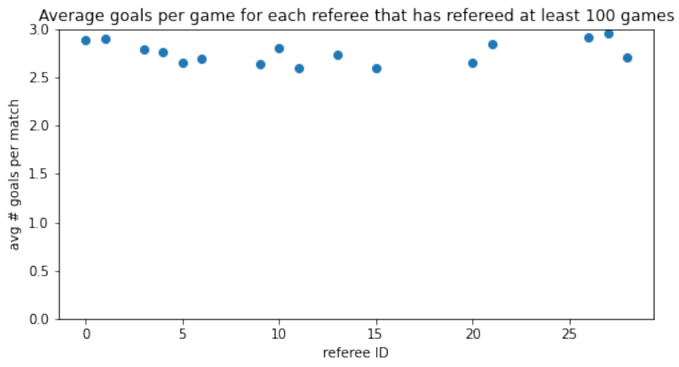

Figure 3: Average number of goals scored per referee. Only referees that have managed at least 100 PL games are included.

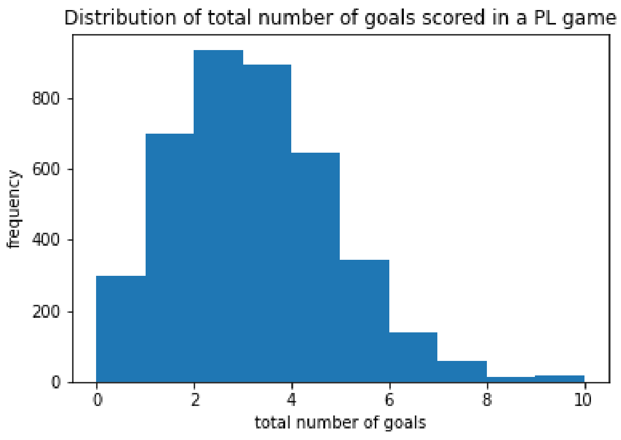

Figure 4: Distribution of number of goals

3 Methodology

3.1 Data Preprocessing

We are only interested in Premier League games, so in the competitions table, I find that the code for these games is "GB1". I filter the games table to only include games with this as its competition_id.

The dataset is regularly updated and contains no empty values so there was no need to remove empty values. However, I made some modifications the games data to make it more more suitable for data analysis. For example, I made the date column's data type into "datetime64[ns]" and converted the referee column into a categorical value. I also wanted to calculate the number of clean sheets[1] so I wrote a method that calculates this for both sides (this can be found in the iPython Notebook). I thought the matchday field might play a factor in how a team performs as well, so I cleaned this column by trimming out any words and whitespace. I have also included a separate column that indicates just the day of week (ranging from 0 to 6). Finally, since the prediction to be made is the result of the game, I created a column called result which will be the target model in our training algorithms. The target is either a 1, 0, or -1 depending corresponding to a win for the home team, a draw, or a win for the away team.

The following shows just a snippet of some of the data pre-processing performed.

```
games.loc[:, "date"] = games["date"].astype('datetime64[ns]')
games["referee"] = games["referee"].astype('category').cat.codes
games["day_code"] = games["date"].dt.dayofweek
games["matchday"] = games["round"].map(lambda x: x.split(".")[0])
games["matchday"] = games["matchday"].astype('int8')
player_valuations.loc[:, "date"] = player_valuations["date"].astype('
                                     datetime64[ns]')
```

The following code snippet shows how I added the target feature result to the games DataFrame.

```
conditions = [games.home_club_goals.eq(games.away_club_goals), games.
                                     home_club_goals.gt(games.
                                     away_club_goals), games.
                                     home_club_goals.lt(games.
                                     away_club_goals)]
choices = [0, 1, -1]
games["result"] = np.select(conditions, choices)
```

3.2 Implementation

3.2.1 Part one

I first split the data into a train set, and a test set, using 20% data for testing.

I initially opt for a Random Forest Classifier provided by sklearn for experimenting purposes. This is because the model requires little configuration and I can gain an initial idea of how much influence each predictor has on the result. Aandom forests build large decision trees, taking a different part of the dataset as the training set for each trew and can give high accuracies quickly.

Further, for this stage, I treat a draw and a loss for the home team as the same. This reduces the problem to a simple binary classification task. Each result is either a win or not a win (1 or 0).

[1]A team keeps a clean sheet in a game of football when they don't conceed.

Predictor removed	accuracy	precision
referee	0.700	0.700
attendance	0.713	0.712
matchday	0.704	0.704
day_code	0.712	0.711

Table 1: Accuracies and precisions with one predictor removed

In our data, the club_id, although a numerical value, is not linearly related to any of the other variables, and hence classification using logistic regression wouldn't be suitable.

I trained a RandomForestClassifier (provided by sklearn.ensemble) on the train set with these columns being the predictors: home_club_id, away_club_id, home_club_position, away_club_position, attendance, referee, matchday, day_code. A day_code of 0 corresponds to a Monday, and 6 to a Sunday. After making predictions on the test set, we already obtain an accuracy of 0.700, and precision of 0.700 which are relatively high. I suspected that the positions in the table of each side had a large influence on the model. Upon removing these two columns and training again, the accuracy and precision drop to 0.608 and 0.605 respectively, which are significant drops. So it is clear that the relative positions of the clubs in the table is playing a big part in the training process. This is not truly realistic on the first day of the season when clubs are ranked alphabetically.

I then included these 2 columns back in and wanted to test the effects of each of attendance, referee, day_code, and matchday on the result. Table 1 shows the accuracies and precisions with one of the columns removed. It is clear that these factors have a very negligible effect on the scores. However, I continue using them as predictors since they could be useful values when training a more complex model. The hyperparameters I used are as follows

- n_estimators: 100
- min_samples_split : 10
- max_features : “sqrt”.

3.2.2 Part two

Next, I change the target column so that there are three classes instead of just two, as discussed in section 1.2. I also include my two custom attributes that are the number of clean sheets each side has kept in their previous 5 games. I use AutoGluon's TabularPredictor to produce models that predict the game result. The best model is a WeightedEnsemble_L2 which gave an accuracy score of 0.593. The precision, calculated using a weighted average again, is 0.555, which is lower than the precision obtained previously. However, This was to be expected, since there are now three classes as opposed to just 2 previously, so there is more room for misclassification. I decided to fit these models again, this time without the clean sheet predictors, and the resulting accuracy and precision are 0.597 and 0.567 respectively. This suggests that these predictors had a slight negative, if any, impact on the model.

For the eval_metric, I use log_loss because this is a classification problem. Other hyperparameters I used are as follows

- preset: best_quality

- time_limit : 600
- eval_metric : log_loss

3.3 Refinements

I experimented more with the RandomForestClassifier. The following is a range of hyper-parameters I used:

- min_samples_split: 10, 20, 30
- n_estimators: 100, 1000, 5000
- max_features: “log2”, “sqrt”

```
X_train, X_test, y_train, y_test = train_test_split(games[predictors],
                                        games["result"], test_size=0.20,
                                        random_state=1)

estimators = [100, 1000, 5000]
min_samples_splits = [10, 20, 30]
max_features = ["sqrt", "log2"]

for num_estimators in estimators:
    for min_samples_split in min_samples_splits:
        for feature_type in max_features:

            print(f"===Hyperparameters used n_estimators: {num_estimators},
                                                min_samples_split: {
                                               min_samples_split},
                                               max_features: {
                                               feature_type}===")
            rf = RandomForestClassifier(n_estimators=num_estimators,
                                               min_samples_split=
                                               min_samples_split,
                                               random_state=1,
                                               max_features=feature_type
                                               )

            rf.fit(X_train, y_train)
            preds = rf.predict(X_test[predictors])

            print(f"Accuracy: {accuracy_score(y_test, preds):.3f}")
            print(f"Precision: {precision_score(y_test, preds, average='
                                               weighted'):.3f}")
```

However, I was consistently obtaining accuracies and precisions of around 0.58 and 0.54 respectively, which isn’t very interesting. I also tried varying the number of games I am considering when looking at clean sheets in previous games, but this also made no difference. This suggests that the limit has effectively been reached with the predictions that can be made using this data.

As one final attempt to improve the predictions, I incorporate average player valuation of each team during a game. I would imagine teams with higher average player price would be more likely to win. This should give us an indication whether the amount a club spends on a players results in any effect on the number of games won. One problem I encountered was that some players don’t have any data valuation data about them. So I skipped over

n_estimators	min_samples_split	max_features	Accuracy	Precision
100	10	sqrt	0.607	0.583
100	10	log2	0.607	0.583
100	20	sqrt	0.595	0.550
100	20	log2	0.595	0.550
100	30	sqrt	0.601	0.558
100	30	log2	0.601	0.558
1000	10	sqrt	0.602	0.575
1000	10	log2	0.602	0.575
1000	20	sqrt	0.598	0.566
1000	20	log2	0.598	0.566
1000	30	sqrt	0.602	0.574
1000	30	log2	0.602	0.574
5000	10	sqrt	0.607	0.589
5000	10	log2	0.607	0.589
5000	20	sqrt	0.604	0.577
5000	20	log2	0.604	0.577
5000	30	sqrt	0.601	0.580
5000	30	log2	0.601	0.580

Table 2: Hyperparameter tuning on RandomForestClassifier

these players when calculating the average. Another problem was that for games before 2014, there is no data about which players played in each match. So I filter out these games in the training and testing datasets.

I kept all the previous predictors as well. The results obtained with using 1000 estimators, minimum_sample_split of 10, and sqrt for max_features are an accuracy of 0.595, and a precision of 0.567. This slightly better than before, but not significantly. This hints that the average value of a player in a team has a very small impact on the chances of the team winning a game.

4 Results

4.1 Model Evaluation

I performed hyperparameter tuning using all 12 predictors that I have mentioned so far, Here is a table of results obtained.

Ultimately, the set of hyperparameters that performed the best was 5000 estimators, min_samples_split of 10. The accuracy is just over 0.6, meaning out of every 5 predictions, about 3 are correct on average. This is better than flipping a coin, and certainly better than guessing (only 0.33 chance of guessing the outcome correctly). The max_features attribute had no effect on the model score, hence can be ignored.

Also using the classification models trained in [3] as a benchmark, they obtain accuracy of 0.511. So my model performs slightly better than that one in terms of accuracy. The betting odds model used by Bet365 obtain 0.521 accuracy [3] (also see section 5.2).

4.2 Justification

The above mentioned set of parameters gave the best precision score and accuracy score. As of writing, today is March 5, 2023, and Manchester United will be playing Liverpool at 4:30pm. I aim to use my trained model to predict the outcome. These are the parameters needed for this particular match

- home_club_id:31 (Liverpool)
- away_club_id:985 (Manchester United)
- home_club_position : 6
- away_club_position : 3
- Referee is Andrew Madley. This corresponds to the id of 1
- matchday: 26
- day_code: 6
- attendance: 52000
- home_avg_player_valuation : 41.6 million euros
- away_avg_player_valuation : 28.9 million euros
- home_clean_sheets_prev_5_games: 4
- away_clean_sheets_prev_5_games: 2

The attendance has been estimated from previous Liverpool games. Predicting the outcome of this game using my trained model gives an output of 1. This means that my model predicts a win for the home team, Liverpool. The actual outcome of the game was a massive win 7-0 for Liverpool.

5 Conclusion

5.1 Reflection

Ultimately, this dataset proved to be a challenge to work with to predict results of Premier League games. In hindsight, 4000 games is not enough data to train an accurate model. It would be interesting to see how these methods I've applied would fare in other top-flight leagues such as La Liga (Spain), Ligue 1 (France), or Serie A (Italy). The high variance of football scores and number of goals scored makes this task particularly challenging.

The final model (weighted average) precision was 0.58 which indicates on average the out of every game that is predicted as a win for the home team roughly 58% are actually wins. And similar for the other labels. Using "micro" averaging technique for calculating precision gives 0.60. This indicates that the model is slightly better at predicting wins/losses compared to draws. This is because out of all the games only a quarter of them are draws, which is less than an even split between three categories.

5.2 Improvement

For further work, it would be a good idea to predict the score of a game rather than just which team wins or if it's a draw.

It would also be a good idea to incorporate modern statistics used in the game such as xG, which is expected goals. For this, it is a good idea to use a different dataset than the one I used since it doesn't have this type of information. [3] trains classification models using xG data and also ELO ratings.

There are lots of other factors that can determine the outcome of a game such as the weather, the player's moods, the importance of particular games. All these factors require more work to incorporate and are definitely avenues for future study.

References

[1] David Cariboo. Football Data from Transfermarkt.

[2] Matthew Henry. What is behind premier league goal rush? *BBC Sport*, Oct 2020.

[3] Corentin Herbinet. Predicting football results using machine learning techniques. *MEng thesis, Imperial College London*, 2018.