Predicting Football Results With Statistical Modelling

Combining the world's most popular sport with everyone's favourite discrete probability distribution, this post predicts football matches using the Poisson distribution.

Switch to R version

David Sheehan
Data scientist interested in sports, politics and Simpsons references

Follow

Football (or soccer to my American readers) is full of clichés: "It's a game of two halves", "taking it one game at a time" and "Liverpool have failed to win the Premier League". You're less likely to hear "Treating the number of goals scored by each team as independent Poisson processes, statistical modelling suggests that the home team have a 60% chance of winning today". But this is actually a bit of cliché too (it has been discussed here, here, here, here and particularly well here). As we'll discover, a simple Poisson model is, well, overly simplistic. But it's a good starting point and a nice intuitive way to learn about statistical modelling. So, if you came here looking to make money, I hear this guy makes £5000 per month without leaving the house.

Poisson Distribution

The model is founded on the number of goals scored/conceded by each team. Teams that have been higher scorers in the past have a greater likelihood of scoring goals in the future. We'll import all match results from the recently concluded Premier League (2016/17) season. There's various sources for this data out there (kaggle, football-data.co.uk, github, API). I built an R wrapper for that API, but I'll go the csv route this time around.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn
from scipy.stats import poisson,skellam

epl_1617 = pd.read_csv("http://www.football-data.co.uk/mmz4281/1617/E0.csv")
```

```
epl_1617 = epl_1617[['HomeTeam','AwayTeam','FTHG','FTAG']]
epl_1617 = epl_1617.rename(columns={'FTHG': 'HomeGoals', 'FTAG': 'AwayGoals'})
epl_1617.head()
```

	HomeTeam	AwayTeam	HomeGoals	AwayGoals
0	Burnley	Swansea	0	1
1	Crystal Palace	West Brom	0	1
2	Everton	Tottenham	1	1
3	Hull	Leicester	2	1
4	Man City	Sunderland	2	1

We imported a csv as a pandas dataframe, which contains various information for each of the 380 EPL games in the 2016-17 English Premier League season. We restricted the dataframe to the columns in which we're interested (specifically, team names and numer of goals scored by each team). I'll omit most of the code that produces the graphs in this post. But don't worry, you can find that code on my github page. Our task is to model the final round of fixtures in the season, so we must remove the last 10 rows (each gameweek consists of 10 matches).

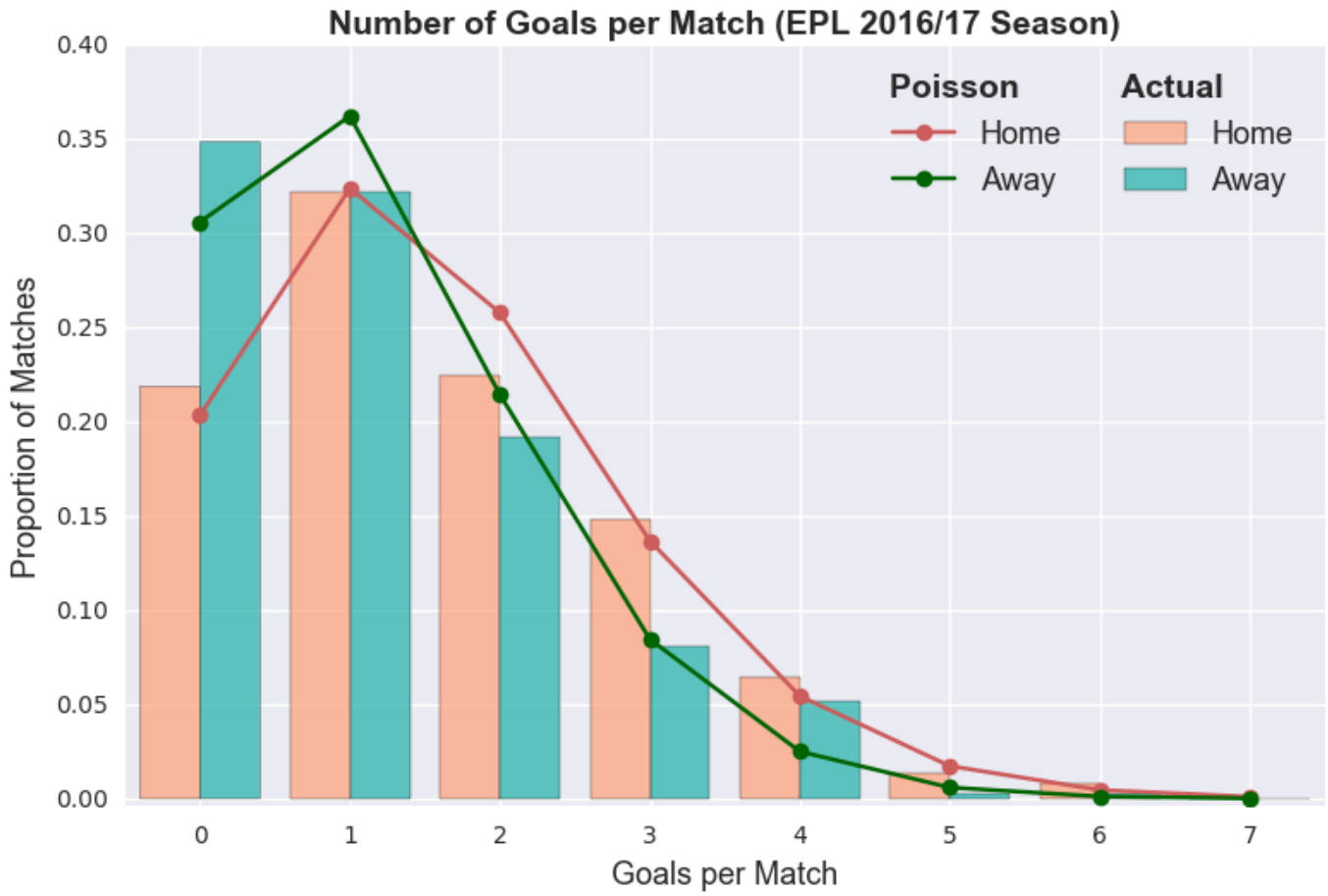
```
epl_1617 = epl_1617[:-10]
epl_1617.mean()
```

```
HomeGoals    1.591892
AwayGoals    1.183784
dtype: float64
```

You'll notice that, on average, the home team scores more goals than the away team. This is the so called 'home (field) advantage' (discussed here) and isn't specific to soccer. This is a convenient time to introduce the Poisson distribution. It's a discrete probability distribution that describes the probability of the number of events within a specific time period (e.g 90 mins) with a known average rate of occurrence. A key assumption is that the number of events is independent of time. In our context, this means that goals don't become more/less likely by the number of goals already scored in the match. Instead, the number of goals is expressed purely as function an average rate of goals. If that was unclear, maybe this mathematical formulation will make clearer:

$$P(x) = \frac{e^{-\lambda}\lambda^{x}}{x!}, \lambda > 0$$

λ represents the average rate (e.g. average number of goals, average number of letters you receive, etc.). So, we can treat the number of goals scored by the home and away team as two independent Poisson distributions. The plot below shows the proportion of goals scored compared to the number of goals estimated by the corresponding Poisson distributions.



We can use this statistical model to estimate the probability of specfic events.

$$\begin{aligned} P(\geq 2|Home) &= P(2|Home) + P(3|Home) + \ldots \\ &= 0.258 + 0.137 + \ldots \\ &= 0.47 \end{aligned}$$

The probability of a draw is simply the sum of the events where the two teams score the same amount of goals.

$$\begin{aligned} P(Draw) &= P(0|Home) \times P(0|Away) + P(1|Home) \times P(1|Away) + \ldots \\ &= 0.203 \times 0.306 + 0.324 \times 0.362 + \ldots \\ &= 0.248 \end{aligned}$$

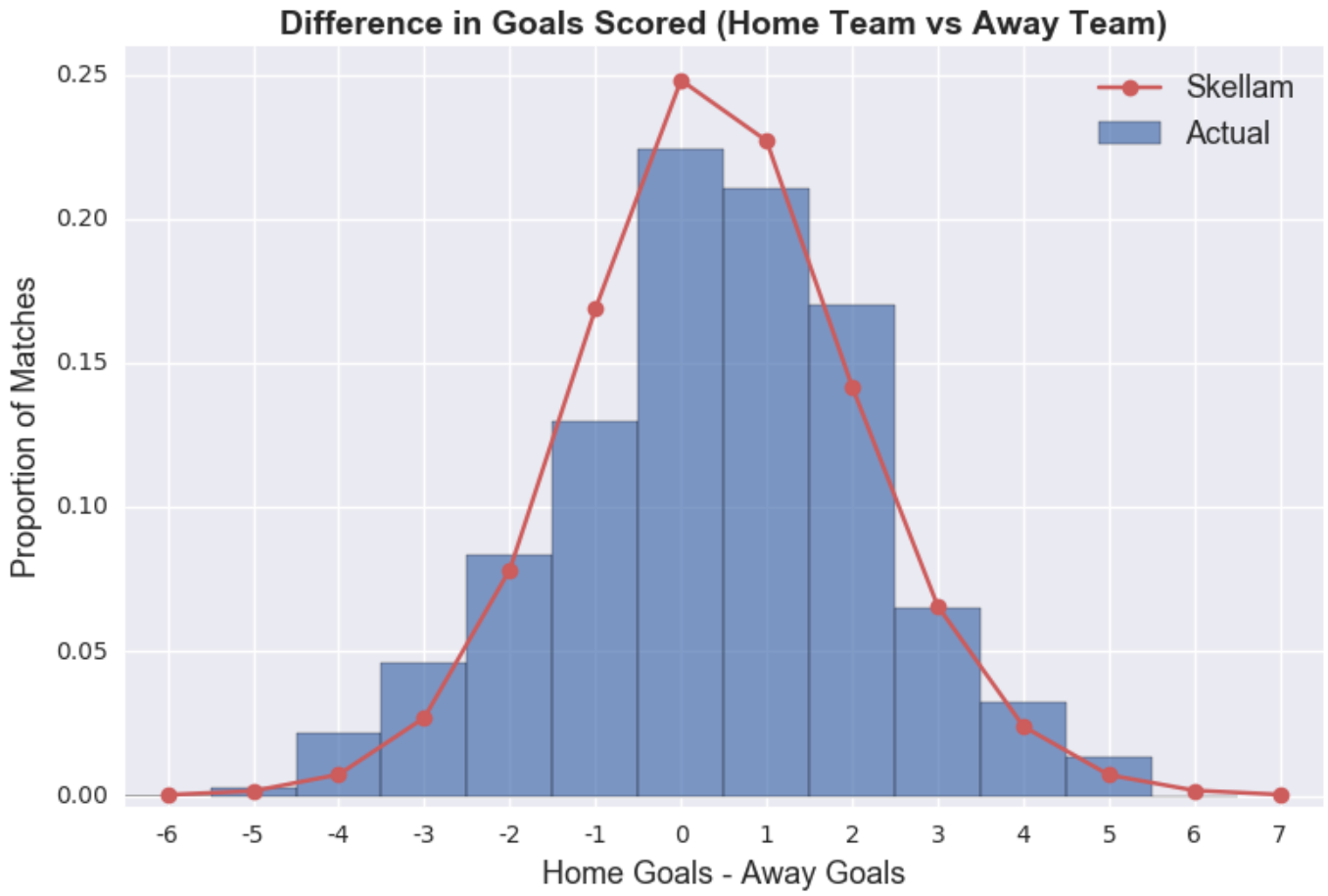
Note that we consider the number of goals scored by each team to be independent events (i.e. P(A n B) = P(A) P(B)). The difference of two Poisson distribution is actually called a Skellam distribution. So we can calculate the probability of a draw by inputting the mean goal values into this distribution.

```
# probability of draw between home and away team
skellam.pmf(0.0,  epl_1617.mean()[0],  epl_1617.mean()[1])
```

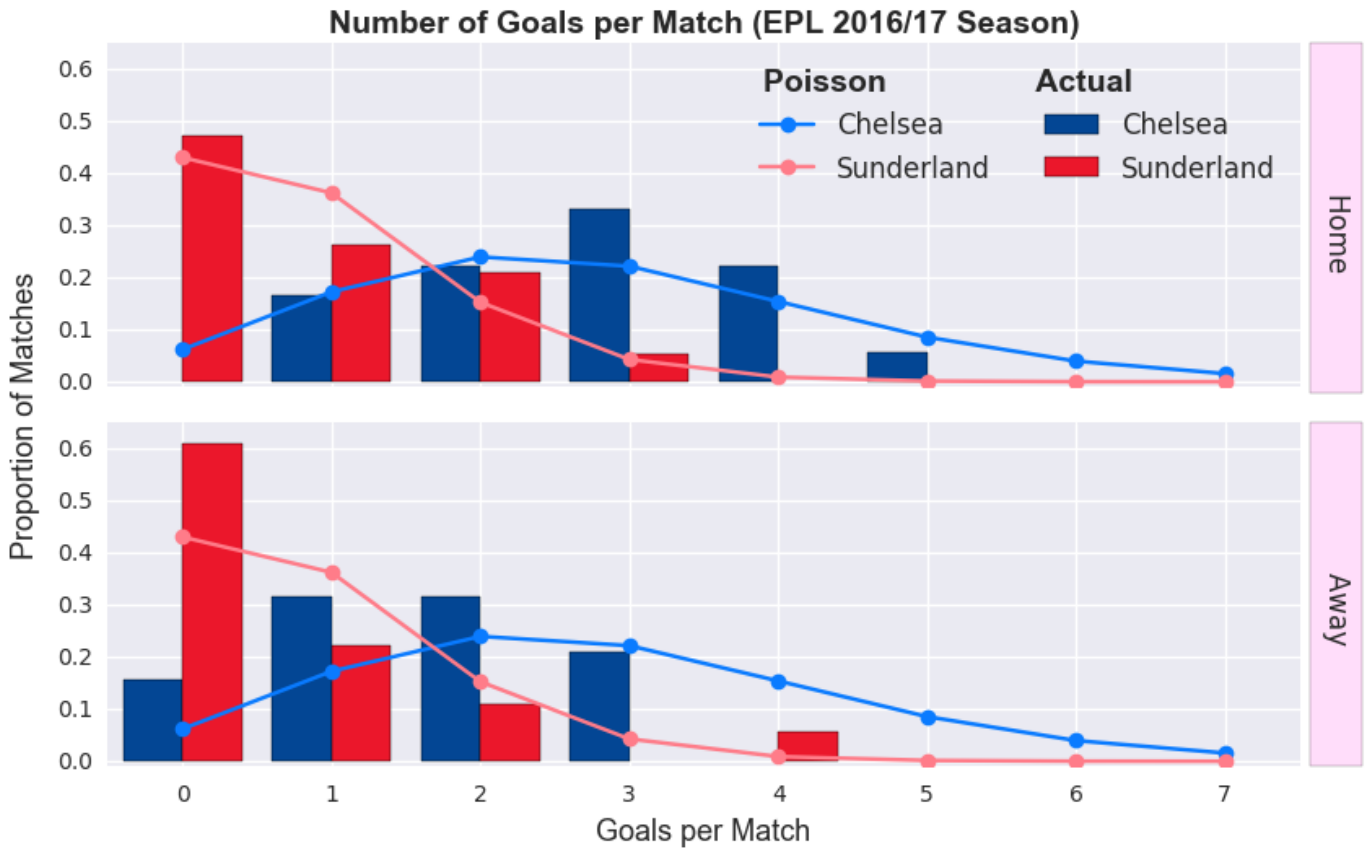
```
0.24809376810717076
```

```
# probability of home team winning by one goal
skellam.pmf(1,  epl_1617.mean()[0],  epl_1617.mean()[1])
```

```
0.22558259663675409
```



So, hopefully you can see how we can adapt this approach to model specific matches. We just need to know the average number of goals scored by each team and feed this data into a Poisson model. Let's have a look at the distribution of goals scored by Chelsea and Sunderland (teams who finished 1st and last, respectively).



Building A Model

You should now be convinced that the number of goals scored by each team can be approximated by a Poisson distribution. Due to a relatively sample size (each team plays at most 19 home/away games), the accuracy of this approximation can vary significantly (especially earlier in the season when teams have played fewer games). Similar to before, we could now calculate the probability of various events in this Chelsea Sunderland match. But rather than treat each match separately, we'll build a more general Poisson regression model (what is that?).

```
# importing the tools required for the Poisson regression model
import statsmodels.api as sm
import statsmodels.formula.api as smf

goal_model_data = pd.concat([epl_1617[['HomeTeam','AwayTeam','HomeGoals']].assign(home=1).rename(
            columns={'HomeTeam':'team', 'AwayTeam':'opponent','HomeGoals':'goals'}),
           epl_1617[['AwayTeam','HomeTeam','AwayGoals']].assign(home=0).rename(
            columns={'AwayTeam':'team', 'HomeTeam':'opponent','AwayGoals':'goals'})])

poisson_model = smf.glm(formula="goals ~ home + team + opponent", data=goal_model_data,
                        family=sm.families.Poisson()).fit()
poisson_model.summary()
```

Dep. Variable:	goals	**No. Observations:**	740

Model:	GLM	**Df Residuals:**	700
Model Family:	Poisson	**Df Model:**	39
Link Function:	log	**Scale:**	1.0
Method:	IRLS	**Log-Likelihood:**	-1042.4
Date:	Sat, 10 Jun 2017	**Deviance:**	776.11
Time:	11:17:38	**Pearson chi2:**	659.
No. Iterations:	8		

Generalized Linear Model Regression Results

	coef	std err	z	P>|z|	[95.0% Conf. Int.]
Intercept	0.3725	0.198	1.880	0.060	-0.016 0.761
team[T.Bournemouth]	-0.2891	0.179	-1.612	0.107	-0.641 0.062
team[T.Burnley]	-0.6458	0.200	-3.230	0.001	-1.038 -0.254
team[T.Chelsea]	0.0789	0.162	0.488	0.626	-0.238 0.396
team[T.Crystal Palace]	-0.3865	0.183	-2.107	0.035	-0.746 -0.027
team[T.Everton]	-0.2008	0.173	-1.161	0.246	-0.540 0.138
team[T.Hull]	-0.7006	0.204	-3.441	0.001	-1.100 -0.302
team[T.Leicester]	-0.4204	0.187	-2.249	0.025	-0.787 -0.054
team[T.Liverpool]	0.0162	0.164	0.099	0.921	-0.306 0.338
team[T.Man City]	0.0117	0.164	0.072	0.943	-0.310 0.334
team[T.Man United]	-0.3572	0.181	-1.971	0.049	-0.713 -0.002
team[T.Middlesbrough]	-1.0087	0.225	-4.481	0.000	-1.450 -0.568
team[T.Southampton]	-0.5804	0.195	-2.976	0.003	-0.963 -0.198
team[T.Stoke]	-0.6082	0.197	-3.094	0.002	-0.994 -0.223
team[T.Sunderland]	-0.9619	0.222	-4.329	0.000	-1.397 -0.526
team[T.Swansea]	-0.5136	0.192	-2.673	0.008	-0.890 -0.137
team[T.Tottenham]	0.0532	0.162	0.328	0.743	-0.265 0.371
team[T.Watford]	-0.5969	0.197	-3.035	0.002	-0.982 -0.211
team[T.West Brom]	-0.5567	0.194	-2.876	0.004	-0.936 -0.177
team[T.West Ham]	-0.4802	0.189	-2.535	0.011	-0.851 -0.109
opponent[T.Bournemouth]	0.4109	0.196	2.092	0.036	0.026 0.796
opponent[T.Burnley]	0.1657	0.206	0.806	0.420	-0.237 0.569
opponent[T.Chelsea]	-0.3036	0.234	-1.298	0.194	-0.762 0.155
opponent[T.Crystal Palace]	0.3287	0.200	1.647	0.100	-0.062 0.720
opponent[T.Everton]	-0.0442	0.218	-0.202	0.840	-0.472 0.384
opponent[T.Hull]	0.4979	0.193	2.585	0.010	0.120 0.875

opponent[T.Leicester]	0.3369	0.199	1.694	0.090	-0.053 0.727
opponent[T.Liverpool]	-0.0374	0.217	-0.172	0.863	-0.463 0.389
opponent[T.Man City]	-0.0993	0.222	-0.448	0.654	-0.534 0.335
opponent[T.Man United]	-0.4220	0.241	-1.754	0.079	-0.894 0.050
opponent[T.Middlesbrough]	0.1196	0.208	0.574	0.566	-0.289 0.528
opponent[T.Southampton]	0.0458	0.211	0.217	0.828	-0.369 0.460
opponent[T.Stoke]	0.2266	0.203	1.115	0.265	-0.172 0.625
opponent[T.Sunderland]	0.3707	0.198	1.876	0.061	-0.017 0.758
opponent[T.Swansea]	0.4336	0.195	2.227	0.026	0.052 0.815
opponent[T.Tottenham]	-0.5431	0.252	-2.156	0.031	-1.037 -0.049
opponent[T.Watford]	0.3533	0.198	1.782	0.075	-0.035 0.742
opponent[T.West Brom]	0.0970	0.209	0.463	0.643	-0.313 0.507
opponent[T.West Ham]	0.3485	0.198	1.758	0.079	-0.040 0.737
home	0.2969	0.063	4.702	0.000	0.173 0.421

If you're curious about the `smf.glm(...)` part, you can find more information here (edit: earlier versions of this post had erroneously employed a Generalised Estimating Equation (GEE)- what's the difference?). I'm more interested in the values presented in the `coef` column in the model summary table, which are analogous to the slopes in linear regression. Similar to logistic regression, we take the exponent of the parameter values. A positive value implies more goals ($e^x > 1 \forall x > 0$), while values closer to zero represent more neutral effects ($e^0 = 1$). Towards the bottom of the table you might notice that `home` has a `coef` of 0.2969. This captures the fact that home teams generally score more goals than the away team (specifically, $e^{0.2969}$ =1.35 times more likely). But not all teams are created equal. Chelsea has a `coef` of 0.0789, while the corresponding value for Sunderland is -0.9619 (sort of saying Chelsea (Sunderland) are better (much worse!) scorers than average). Finally, the `opponent*` values penalize/reward teams based on the quality of the opposition. This relfects the defensive strength of each team (Chelsea: -0.3036; Sunderland: 0.3707). In other words, you're less likely to score against Chelsea. Hopefully, that all makes both statistical and intuitive sense.

Let's start making some predictions for the upcoming matches. We simply pass our teams into `poisson_model` and it'll return the expected average number of goals for that team (we need to run it twice- we calculate the expected average number of goals for each team separately). So let's see how many goals we expect Chelsea and Sunderland to score.

```
poisson_model.predict(pd.DataFrame(data={'team': 'Chelsea', 'opponent': 'Sunderland',
                                         'home':1},index=[1]))
```

```
array([ 3.06166192])
```

```
poisson_model.predict(pd.DataFrame(data={'team': 'Sunderland', 'opponent': 'Chelsea',
                                         'home':0},index=[1]))
```

```
array([ 0.40937279])
```

Just like before, we have two Poisson distributions. From this, we can calculate the probability of various events. I'll wrap this in a `simulate_match` function.

```
def simulate_match(foot_model, homeTeam, awayTeam, max_goals=10):
    home_goals_avg = foot_model.predict(pd.DataFrame(data={'team': homeTeam,
                                                           'opponent': awayTeam,'home':1},
                                                     index=[1])).values[0]
    away_goals_avg = foot_model.predict(pd.DataFrame(data={'team': awayTeam,
                                                           'opponent': homeTeam,'home':0},
                                                     index=[1])).values[0]
    team_pred = [[poisson.pmf(i, team_avg) for i in range(0, max_goals+1)] for team_avg in [home_g
    return(np.outer(np.array(team_pred[0]), np.array(team_pred[1])))
simulate_match(poisson_model, 'Chelsea', 'Sunderland', max_goals=3)
```

```
array([[ 0.03108485,  0.01272529,  0.00260469,  0.00035543],
       [ 0.0951713 ,  0.03896054,  0.00797469,  0.00108821],
       [ 0.14569118,  0.059642  ,  0.01220791,  0.00166586],
       [ 0.14868571,  0.06086788,  0.01245883,  0.0017001 ]])
```

This matrix simply shows the probability of Chelsea (rows of the matrix) and Sunderland (matrix columns) scoring a specific number of goals. For example, along the diagonal, both teams score the same the number of goals (e.g. P(0-0)=0.031). So, you can calculate the odds of draw by summing all the diagonal entries. Everything below the diagonal represents a Chelsea victory (e.g P(3-0)=0.149), And you can estimate P(Over 2.5 goals) by summing all entries except the four values in the upper left corner. Luckily, we can use basic matrix manipulation functions to perform these calculations.

```
chel_sun = simulate_match(poisson_model, "Chelsea", "Sunderland", max_goals=10)
# chelsea win
np.sum(np.tril(chel_sun, -1))
```

```
0.8885986612364134
```

```
# draw
np.sum(np.diag(chel_sun))
```

```
0.084093492686495977
```

```
# sunderland win
np.sum(np.triu(chel_sun, 1))
```

```
0.026961819942853051
```

Hmm, our model gives Sunderland a 2.7% chance of winning. But is that right? To assess the accuracy of the predictions, we'll compare the probabilities returned by our model against the odds offered by the Betfair exchange.

Sports Betting/Trading

Unlike traditional bookmakers, on betting exchanges (and Betfair isn't the only one- it's just the biggest), you bet against other people (with Betfair taking a commission on winnings). It acts as a sort of stock market for sports events. And, like a stock market, due to the efficient market hypothesis, the prices available at Betfair reflect the true price/odds of those events happening (in theory anyway). Below, I've posted a screenshot of the Betfair exchange on Sunday 21st May (a few hours before those matches started).

Sun 21 May — Multiples

Coming up			1		X		2	
Today 15:00	Arsenal Everton	Matched: GBP 502,864	1.4 £2633	1.41 £584	5.7 £611	5.8 £132	8.6 £351	8.8 £440
Today 15:00	Burnley West Ham	Matched: GBP 127,063	2.38 £119	2.4 £84	3.6 £245	3.65 £2403	3.25 £1601	3.3 £473
Today 15:00	Chelsea Sunderland	Matched: GBP 312,295	1.13 £60703	1.14 £11594	11.5 £334	12 £221	29 £284	30 £23
Today 15:00	Hull Tottenham	Matched: GBP 274,460	9.2 £40	9.4 £784	5.8 £790	5.9 £642	1.39 £2224	1.4 £5137
Today 15:00	Leicester Bournemouth	Matched: GBP 216,678	1.87 £320	1.89 £170	4.1 £951	4.3 £1557	4.3 £279	4.4 £434
Today 15:00	Liverpool Middlesbrough	Matched: GBP 857,747	1.14 £176274	1.15 £56345	10.5 £2988	11 £2701	28 £1807	29 £1581
Today 15:00	Man Utd C Palace	Matched: GBP 684,142	2.4 £2992	2.42 £9765	3.45 £4522	3.5 £658	3.35 £696	3.4 £1223
Today 15:00	Southampton Stoke	Matched: GBP 100,032	1.75 £448	1.76 £2566	4.1 £1079	4.2 £49	5.2 £74	5.3 £217
Today 15:00	Swansea West Brom	Matched: GBP 118,771	2.32 £7	2.34 £825	3.5 £1942	3.55 £304	3.45 £612	3.5 £76
Today 15:00	Watford Man City	Matched: GBP 589,601	19.5 £27	20 £101	9.8 £203	10 £262	1.17 £32750	1.18 £3609

The numbers inside the boxes represent the best available prices and the amount available at those prices. The blue boxes signify back bets (i.e. betting that an event will happen- going long using stock market terminology), while the pink boxes represent lay bets (i.e. betting that something won't happen- i.e. shorting). For example, if we were to bet £100 on Chelsea to win, we would receive the original amount plus 100*1.13= £13 should they win (of course, we would lose our £100 if they didn't win). Now, how can we compare these prices to the probabilities returned by our model? Well, decimal odds can be converted to the probabilities quite easily: it's simply the inverse of the decimal odds. For example, the implied probability of Chelsea winning is 1/1.13 (=0.885- our model put the probability at 0.889). I'm focusing on decimal odds, but you might also be familiar with Moneyline (American) Odds (e.g. +200) and fractional odds (e.g. 2/1). The relationship between decimal odds, moneyline and probability is illustrated in the table below. I'll stick with decimal odds because the alternatives are either unfamiliar to me (Moneyline) or just stupid (fractional odds).

Convert to Decimal Odds

Match	Home	Draw	Away
Arsenal v Everton	71.4 %	17.5 %	11.6 %
Burnley v West Ham	42 %	27.8 %	30.8 %
Chelsea v Sunderland	88.5 %	8.7 %	3.4 %
Hull v Tottenham	10.9 %	17.2 %	71.9 %
Leicester v Bournemouth	53.5 %	24.4 %	23.3 %

Match	Home	Draw	Away
Liverpool v Middlesbrough	87.7 %	9.5 %	3.6 %
Man Utd v C Palace	41.7 %	29 %	29.9 %
Southampton v Stoke	57.1 %	24.4 %	19.2 %
Swansea v West Brom	43.1 %	28.6 %	29 %
Watford v Man City	5.1 %	10.2 %	85.5 %

Chance of Occurence (EPL Fixtures 21st May 2017)

Source: Betfair Exchange

So, we have our model probabilities and (if we trust the exchange) we know the true probabilities of each event happening. Ideally, our model would identify situations the market has underestimated the chances of an event occurring (or not occurring in the case of lay bets). For example, in a simple coin toss game, imagine if you were offered $2 for every $1 wagered (plus your stake), if you guessed correctly. The implied probability is 0.333, but any valid model would return a probability of 0.5. The odds returned by our model and the Betfair exchange are compared in the table below.

Match		Home	Draw	Away
	Betfair	0.714	0.175	0.116
Arsenal v Everton	Predicted	0.533	0.226	0.241
	Difference	0.181	0.051	0.125
	Betfair	0.42	0.278	0.308
Burnley v West Ham	Predicted	0.461	0.263	0.276
	Difference	0.041	0.015	0.032
	Betfair	0.885	0.087	0.034
Chelsea v Sunderland	Predicted	0.889	0.084	0.027
	Difference	0.004	0.003	0.007
	Betfair	0.109	0.172	0.719
Hull v Tottenham	Predicted	0.063	0.138	0.799
	Difference	0.046	0.034	0.08
	Betfair	0.535	0.244	0.233
Leicester v Bournemouth	Predicted	0.475	0.22	0.306
	Difference	0.06	0.024	0.073
	Betfair	0.877	0.095	0.036
Liverpool v Middlesbrough	Predicted	0.77	0.161	0.069
	Difference	0.107	0.066	0.033
Man Utd v C Palace	Betfair	0.417	0.29	0.299

Match		Home	Draw	Away
	Predicted	0.672	0.209	0.119
	Difference	0.255	0.081	0.18
	Betfair	0.571	0.244	0.192
Southampton v Stoke	Predicted	0.496	0.277	0.226
	Difference	0.075	0.033	0.034
	Betfair	0.431	0.286	0.29
Swansea v West Brom	Predicted	0.368	0.266	0.366
	Difference	0.063	0.02	0.076
	Betfair	0.051	0.102	0.855
Watford v Man City	Predicted	0.167	0.203	0.631
	Difference	0.116	0.101	0.224

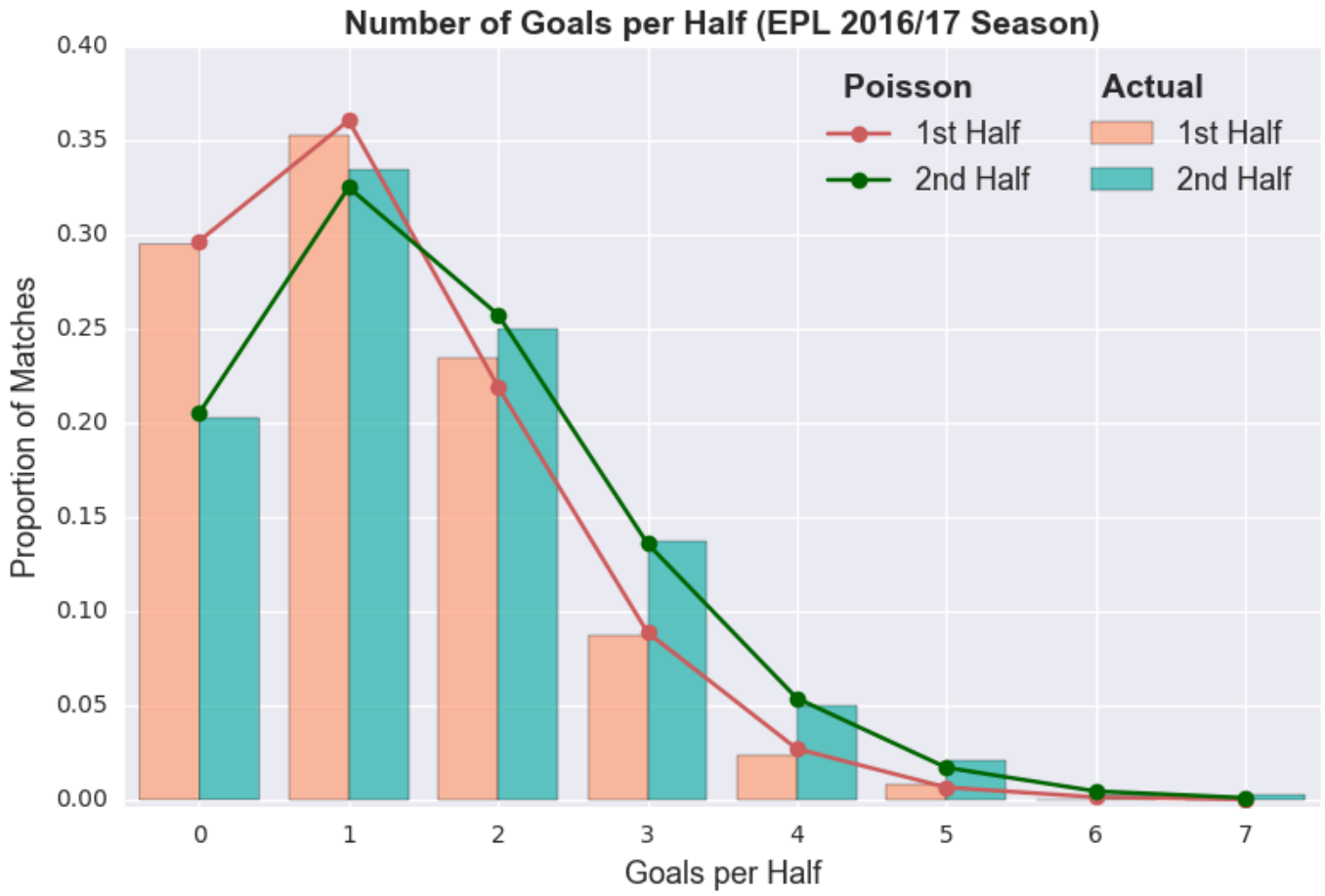
Green cells illustrate opportunities to make profitable bets, according to our model (the opacity of the cell is determined by the implied difference). I've highlighted the difference between the model and Betfair in absolute terms (the relative difference may be more relevant for any trading strategy). Transparent cells indicate situations where the exchange and our model are in broad agreement. Strong colours imply that either our model is wrong or the exchange is wrong. Given the simplicity of our model, I'd lean towards the latter.

Something's Poissony

So should we bet the house on Manchester United? Probably not (though they did win!). There's some non-statistical reasons to resist backing them. Keen football fans would notice that these matches represent the final gameweek of the season. Most teams have very little to play for, meaning that the matches are less predictable (especially when they involve unmotivated 'bigger' teams). Compounding that, Man United were set to play Ajax in the Europa Final three days later. Man United manager, Jose Mourinho, had even confirmed that he would rest the first team, saving them for the much more important final. In a similar fashion, injuries/suspensions to key players, managerial sackings would render our model inaccurate. Never underestimate the importance of domain knowledge in statistical modelling/machine learning! We could also think of improvements to the model that would incorporate time when considering previous matches (i.e. more recent matches should be weighted more strongly).

Statistically speaking, is a Poisson distribution even appropriate? Our model was founded on the belief that the number goals can be accurately expressed as a Poisson distribution. If that assumption is misguided, then the model outputs will be unreliable. Given a Poisson distribution with mean λ, then the number of events in half that time period follows a Poisson distribution with mean $\lambda/2$. In football terms, according to our Poisson model, there should be an equal number of goals in the first and second halves. Unfortunately, that doesn't appear to hold true.

```
epl_1617_halves = pd.read_csv("http://www.football-data.co.uk/mmz4281/1617/E0.csv")
epl_1617_halves = epl_1617_halves[['FTHG', 'FTAG', 'HTHG', 'HTAG']]
epl_1617_halves['FHgoals'] = epl_1617_halves['HTHG'] + epl_1617_halves['HTAG']
epl_1617_halves['SHgoals'] = epl_1617_halves['FTHG'] + epl_1617_halves['FTAG'] -
                             epl_1617_halves['FHgoals']
epl_1617_halves = epl_1617_halves[['FHgoals', 'SHgoals']]
```



We have irrefutable evidence that violates a fundamental assumption of our model, rendering this whole post as pointless as Sunderland!!! Or we can build on our crude first attempt. Rather than a simple univariate Poisson model, we might have more success with a bivariate Poisson distriubtion. The Weibull distribution has also been proposed as a viable alternative. These might be topics for future blog posts.

Summary

We built a simple Poisson model to predict the results of English Premier League matches. Despite its inherent flaws, it recreates several features that would be a necessity for any predictive football model (home advantage, varying offensive strengths and opposition quality). In conclusion, don't wager the rent money, but it's a good starting point for more sophisticated realistic models. Thanks for reading!

Tags: Betfair epl football Poisson python soccer

Categories: football python

Updated: June 04, 2017

Previous Next

LEAVE A COMMENT

35 Comments https://dashee87.github.io/ 1 Login

Recommend 3 Share Sort by Best

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS

Name

tai man chan • a day ago

it seems the 2017-06-04-predicting-football-results-with-statistical-modelling.ipynb is removed.... :<

• Reply • Share ›

David Sheehan Mod tai man chan • a day ago

Hmm, try this link.

• Reply • Share ›

Marcel Bruckmann • a month ago

Hi David,

I am new to create something in python. Can you write the code to create the bar charts?

It would be very nice.

Thanks!

BR

Marcel

• Reply • Share ›

David Sheehan Mod ↗ Marcel Bruckmann • a month ago

Hey Marcel! You can find the code for that plot in the accompanying Jupyter notebook.

^ | v • Reply • Share ›

David Jägering • 2 months ago

Very nice article.

"And you can estimate P(Over 2.5 goals) by summing all entries except the four values in the upper left corner." -> not entirely correct. I think you missed 2:0 and 0:2.

I append some further calculations.

```
print("Over 2.5 goals")
print(np.sum(array[2:])+np.sum(array[:2,2:])-np.sum(array[2:3,0])-np.sum(array[0:1,2]))
print("Under 2.5 goals")
print(np.sum(array[:2,:2])+array.item((0,2))+array.item((2,0)))
print("Home Clean Sheet Yes")
print(np.sum(array[:,0]))
print("Home Clean Sheet No")
print(np.sum(array[:,1:]))
print("Away Clean Sheet Yes")
print(np.sum(array[:1]))
print("Away Clean Sheet No")
print(np.sum(array[1:]))
print("Both Teams score Yes")
print(np.sum(array[1:,1:]))
print("Both Teams score No")
print(np.sum(array[:1])+np.sum(array[1:,0]))
```

^ | v • Reply • Share ›

David Sheehan Mod ↗ David Jägering • 2 months ago

You're right, David. Good spot! And thanks for the code. I'll add a reference to your comment in the text, as I'm sure it'll be helpful to some people. I should also point out that you can alternatively determine some probabilities by taking the complement of an easier calculation.

```
print("Both Teams score Yes")
print(np.sum(array[1:,1:]))
print("Both Teams score No")
print(1- np.sum(array[1:,1:]))
```

1 ^ | v • Reply • Share ›

Pang Derrick • 4 months ago

Very nice article!
May I know where is the source code of this project for Python version? I only found R version on Github... thanks a lot!

^ | v • Reply • Share ›

David Sheehan Mod ↗ Pang Derrick • 4 months ago

The corresponding iPython notebook is listed in the Jupyter folder in the blogScripts repository (link). If you have Jupyter installed, then you can run the code from there.

There's no python script, as such. You'd need to copy the code you want and paste it into your own .py file.

Reply • Share ›

Moamen Sayed • 5 months ago

Very nice article, thanks for that
i would like to ask why it takes too long to produce the result each time i run the script

Reply • Share ›

David Sheehan Mod → Moamen Sayed • 5 months ago

Hmm, it might be worth finding out which specific part is particularly slow. I suspect it could be the pd.read_csv(url) call, as that's usually the slowest bit for me. If that's the case, you could download the csvs and import them locally.

Reply • Share ›

Billy Yu • 6 months ago

very good article. But if I want to add some other information, such as the result between the two teams, How can I model?

Reply • Share ›

David Sheehan Mod → Billy Yu • 6 months ago

Good question! It probably depends on the data. Say if you wanted to include the result when in their last encounter or number of days since each team last played, you'd simply append those columns to the pandas dataframe or numpy array (whichever way you prefer to work). You'd then feed those new columns into the model formula like this:

poisson_model = smf.glm(formula="goals ~ home + team + opponent + newcol0 + newcol1", data=goal_model_data,
family=sm.families.Poisson()).fit()

I'm actually not too sure how including this information would affect the model. I'll add to my list of things to do on this topic.

Reply • Share ›

Pukie Devina → David Sheehan • 2 months ago

Hi David, first of all thanks for the very nice and clear article.

I have a little issue with the info adding: i used the above structure but i get the following error with any new column i want to use:
"patsy.PatsyError: Error evaluating factor: NameError: name 'Wf' is not defined
goals ~ home + team + opponent + Wf".

The new column "Wf" was added to the csv and treated in the program exactly like the others from the goal_model_data:
"goal_model_data =
pd.concat([epl_1617[['HomeTeam','AwayTeam','HomeGoals','Wf']]".

it's also added here :"epl_1617 =
epl_1617[['HomeTeam','AwayTeam','FTHG','FTAG','Wf']]" and i can visualize it after

after
print(epl_1617.head())

(Error appears in when simulate_match is called)

i get the same error also if use another column from the csv (for ex. "HTHG")

Any ideas where i went wrong? (i checked also some issues at stack overflow, ex :#3987, but couldn't figure it out)

Thanks

• Reply • Share ›

Amir Rezaee • 8 months ago

Hey David,

Very strange but I keep getting this error during the simulate_match function. Any idea why?

KeyError Traceback (most recent call last)
<ipython-input-10-57de75f05810> in <module>()
8 team_pred = [[poisson.pmf(i, team_avg) for i in range(0, max_goals+1)] for team_avg in [home_goals_avg, away_goals_avg]]
9 return(np.outer(np.array(team_pred[0]), np.array(team_pred[1])))
---> 10 simulate_match(poisson_model, 'Chelsea', 'Sunderland', max_goals=3)

<ipython-input-10-57de75f05810> in simulate_match(foot_model, homeTeam, awayTeam, max_goals)
2 home_goals_avg = foot_model.predict(pd.DataFrame(data={'team': homeTeam,
3 'opponent': awayTeam,'home':1},
----> 4 index=[1]))[0]
5 away_goals_avg = foot_model.predict(pd.DataFrame(data={'team': awayTeam,
6 'opponent': homeTeam,'home':0},

see more

• Reply • Share ›

Amir Rezaee ➚ Amir Rezaee • 8 months ago

If I reply [0] after index=[1])), it works - hmmm

• Reply • Share ›

Loudtown Network ➚ Amir Rezaee • 6 months ago

hello, am getting the same error what did you do to fix this error

• Reply • Share ›

David Sheehan Mod ➚ Loudtown Network • 6 months ago

Hey! Apologies for this error. I'm now getting the error when I run it. I suspect it has something to do with the version of pandas you're using (this was initially built on an old one). Anyway, I've added a .values to the function, which should fix the problem for you.

• Reply • Share ›

Joe Devola • 9 months ago

Is it possible to take the date of the games into consideration, to weight recent results heavier?

• Reply • Share ›

David Sheehan Mod ➚ Joe Devola • 9 months ago

Sure. That exact idea was one of the improvements proposed by the Dixon-Cole model. In that paper, they include a non-increasing function to weight more recent matches more strongly. They suggest a negative exponential, exp(-ζ * t), where ζ will determine the extent to which the model will favour recency. Unfortunately, there's no real correct value for ζ. This post explores the sensitivity of the model to this parameter.

• Reply • Share ›

Joe Devola ➚ David Sheehan • 9 months ago

Thanks. Any suggestion for how it would be done with your Python code?

1 • Reply • Share ›

Joe Devola ➚ Joe Devola • 7 months ago

Bump!

• Reply • Share ›

David Sheehan Mod ➚ Joe Devola • 7 months ago

Ah, sorry Joe! I didn't see this. I had a think about how to implement this in Python. Admittedly, I'm more comfortable building these types of models in R. But it looks like you'd have to build a more bespoke model and then try to optimise the parameters using some custom maximum likelihood function (which may well be how smf.glm calculated the coefficients for the model in this post).

Given the popularity of this post, I do intend to follow it up with a more sophisticated approach. I just can't put any timeframe on it. Apologies for the slow, vague, unhelpful reply.

• Reply • Share ›

Joe Devola ➚ David Sheehan • 7 months ago

No problem at all, I'll keep a look out for the follow up then!

• Reply • Share ›

Joseph Dau • 9 months ago

first of all this article is really helpful. i am a noob in machine learning and still in the practice aspect of it. I'm having problems with where the goals_model is coming from in the simulate_match function. Could you explain it to me or anyone who might see this post. thanks.

• Reply • Share ›

David Sheehan Mod ➚ Joseph Dau • 9 months ago

Hey Joseph! Thanks for the comment. Good spot! That's a typo. goals_model should have been foot_model. I think I had used the former in earlier drafts. I've fixed it now. I

hope you didn't spend too long wondering what had gone wrong.

Reply • Share ›

Joseph Dau ➚ David Sheehan • 9 months ago

Cool. Thank you for clearing that up and responding. It makes sense now.

Reply • Share ›

Robert • a year ago

Very nice article.

As a newbie python programmer I was wondering how to create a simple prediction game for my friends.

We all play in a an amateur town league of 10 teams. All games are played on the same pitch (no home - away)
My plan is to create an algorithm that will first calculate each team's strength (for example 0-100) and then probability of a win based on previous matches, goals scored, etc.

Could you guys point me in the right direction please?
How to approach making of such algorithm?

Thank you.

Reply • Share ›

David Sheehan Mod ➚ Robert • a year ago

What you're looking for is not too different from the approach I outlined in this post. To quantify team strength, calculate the average number of goals score per team. Since there's no home and away, each team will have just one score. Plug these values into two Poisson distributions. You could then use Monte Carlo simulations to determine the proportion of times that one team beats another. This link might explain it better than me: https://www.pinnacle.com/en.... The Jupyter notebook for this post might also be a useful starting point: https://github.com/dashee87....

Reply • Share ›

manejandodatos • a year ago

amazing article!

Reply • Share ›

José María Miotto • a year ago

Nice post, however, you are not verifying the model. I mean, instead of saying that the odds are wrong, you should measure if your prediction is better or worse than the odds. You could do that just by using the Brier score, which is a score to evaluate probabilistic predictions. If you do it, please post the result, it would be interesting!

Reply • Share ›

Sash Dev ➚ José María Miotto • a year ago

I'd like to see even more - evaluate expected $. Because you can achieve a positive probabilistic rate, but got a negative balance at the end due coefs like 1.05 etc

Reply • Share ›

David Sheehan Mod ↗ José María Miotto • a year ago

Thanks for the comment! You're right. If I wanted to test the model, I could calculate the Brier score. But given the crudeness of the model, I don't feel it merited the effort that would be needed to test it across enough game weeks. Plus, using the Betfair exchange, I could point out that a good model needs to take into account less quanitative things like team motivations.

^ v • Reply • Share ›

José María Miotto ↗ David Sheehan • a year ago

The Betfair exchange is a reference, but is not clear that those probabilities are the 'true' ones. I mean, it would not be the first time that the market is wrong, and in fact that has to be the natural case. Your model is very simple, and that