EE-621 | Markov Chains and queuing systems

# Monte Carlo Simulations of the English Premier League

*Guided by:*
Prof. Prasanna Chaporkar

*Submitted By :*
Deep Karman Pal Singh | 16D070063

## 1 Objectives

English Premier league is the top league of football in England. It is widely regarded as one of the most competitive leagues in the world. Of interest in this league are question like - how many points does a team need (on average) to win the league, how many to qualify for the UEFA Champions League, for the Europa League and how many points does a team need to avoid relegation. Through this project, I wish to answer the above questions. I also answer that looking at the data from the last 2 years, what are the odds for each team to win the league.

## 2 Why Monte Carlo Methods

A direct computation of the probabilities of the state of the table at the end of the league is not feasible due to the large number of possible states.

Consider, after each match, three cases - wins, loss or a draw. This gives that each state (of the points table) evolves with games, and three possibilities for the next state. Over the 380 games of the league, this means that there are $3^{380}$ possible states, which is too large to compute the probabilities over. Note that here we ignored goal difference to simplify analysis, which would increase the number of states by a large amount. Hence direct computation is not feasible, and we move to Monte Carlo methods.

## 3 Our Approach to the Problem

### 3.1 Simulating the League

We consider 20 teams in the league, as per the **2017-18 season**. In our model we would ignore the form of a team, hence the order in which matches are played is irrelevant. We create a simple schedule for these 20 teams and simulate the season match by match.

## 3.2 Simulating a match

In each match, we consider each team scores goals as per a **Poisson distribution**, whose parameter we estimate as below. Based on this we consider the evolution of the points table. Note that we would need to keep a track of goals to resolve differences in the points table (based on goal difference, and then goals for). We do this for all of the 380 games in the season.

## 3.3 Estimating the Parameters

We assumed that the distribution of goals by a team over many matches is Poisson. We note by empirically plotting the data that this is accurate. Now we wish to estimate these parameters.

We need to about a team's attack strength (at home and away) and defensive strength (at home and away), hence we shall need 4 parameters for each team. We consider the average number of goal a home team scores, and the average an away team scores over the entire league. These form our base parameters. We shall weigh these with respect to the attacks and defences of teams playing to get the Poisson parameters of the team.

To model the attacking strength of a home (away) team, we consider its weight as the factor by which the team scores more home (away) goals on average over the average number of home (away) goals scored in the league.
Similarly, the defensive strength (we actually calculate what could be considered an inverse of the strength, since smaller value implies stronger defense) of a home (away) team is the factor by which average number of goals the home (away) team concedes is more than the average number of goals conceded by home (away) teams.

We now expand on the above explanation with the equations used, say for Manchester City (ManC) versus Liverpool (Liv) match

$$\lambda_{\text{av,H}} = \text{Average number of Home Goals scored in the league}$$
$$\lambda_{\text{ManC,Manc v Liv}} = \lambda_{\text{av,H}} \times AttSrt_{\text{ManC,H}} \times DefStr_{\text{Liv,A}}$$
$$\lambda_{\text{Liv,Manc v Liv}} = \lambda_{\text{av,A}} \times AttSrt_{\text{Liv,A}} \times DefStr_{\text{ManC,H}}$$
$$AttSrt_{\text{ManC,H}} = \text{Attacking Strength of ManC at Home}$$
$$AttSrt_{\text{ManC,H}} = \frac{\text{Average number of goals ManC scores when playing at home}}{\text{Average number of goals scored by home teams}}$$
$$DefStr_{\text{Liv,A}} = \text{Defensive Strength of Liv at Away}$$
$$DefStr_{\text{Liv,A}} = \frac{\text{Average number of goals Liv concedes when playing away}}{\text{Average number of goals conceded by away team}}$$

Note that this gives that an average team shall score and concede an average number of goals. This is consistent since the average of a Poisson distribution is its parameter. Running our model, we see that the results appear to be distributed like a real table, hence lending credibility to our assumptions.

## 3.4 Data Acquisition and Cleanup

We took data from the entire 2017-18 season, as well as from the 2018-19 season. The source is linked here. Merging of the data from two season led to some points we had to consider, listed below.

First, the 2018-19 season is not complete, but ongoing. So we just took the season so far, incorporating all the games. This is uptil games 37 for all teams, only 1 matchday is missing for all.

Next, due to relegation and promotion there is a difference of teams in the two seasons. To solve this, we removed all matched with the three newly promoted teams in the 2018-19 season. Hence we only had 262 matches from the 2018-19 season in our dataset, and all 380 matches from 2017-18 season.

Since the data for the three teams that got relegated was missing from the 2018-19 season, we did not add any bias for the newer season over the old. Note that the data imbalance does not effect the calculations of the attacking and defensive strengths since we only consider averages over games played.

# 4 Observations & Results

We ran two simulations, one only with data from 2017-18 season and one with data from both the seasons. In each case, we ran multiple simulations, over 1000, 10,000, and 1,000,000 iterations.

## 4.1 Only 2017-18 Season's Data

We observe that (for the simulation with 1 million leagues)

- Due to Manchester City's amazing record in the 2017-18 season (total 100 points, highest ever in the league), they won 89.8% of all simulations. Next were Liverpool and Manchester United, winning 4.7% and 3.3% of the simulations respectively. Tottenham won 1.9% of the time, Chelsea 0.2% and Arsenal 0.01%.
- No team other than these (not in the colloquial "Big Six") won the league in any simulation. This only serves to show why Leicester City's 2015-16 season, where they won the league was so exceptional (according to this, rarer than one in a million).
- On average, the second placed team made 85.17 points. Hence, on average, **scoring 86 points would win you the league**.
- To qualify for UEFA Champions League, a team needs to be in the top 4. The 5th placed team made, on average, 69.5 points. Hence to **qualify for Champions League**, a team need, on average, **70 points**.

- Similarly, **for Europa League**, 7th placed team made 57.4 points. Hence to be in the Europa League, a team needs **58 points**, on average.
- The 18th placed team made, on average, 32.89 points. Hence, **to avoid relegation**, a team need **33 points**, on average.

## 4.2 Both 2017-18 and 2018-19 Season's Data

We observe that (for the simulation with 1 million leagues)

- Due to Manchester City's amazing record in the both the seasons, they won 80.8% of all simulations. Next were Liverpool and Tottenham, winning 17.3% and 1.4% of the simulations respectively. Manchester United won 0.23% of the time, Chelsea 0.21% and Arsenal 0.02%.
- No team other than these (not in the colloquial "Big Six") won the league in any simulation. Liverpool won a much larger proportion due to their great record in the 2018-19 season, while the reverse occurred for Manchester United.
- On average, the second placed team made 86.35 points. Hence, on average, **scoring 87 points would win you the league**. This increased due to the great record of two teams, Manchester City and Liverpool over the two seasons.
- To qualify for UEFA Champions League, a team needs to be in the top 4. The 5th placed team made, on average, 68.15 points. Hence to **qualify for Champions League**, a team need, on average, **69 points**. This fell due to a weak performance by all three of Chelsea, Arsenal, and Manchester United in the 2018-19 season.
- Similarly, **for Europa League**, 7th placed team made 58.33 points. Hence to be in the Europa League, a team needs **59 points**, on average.
- The 18th placed team made, on average, 32.65 points. Hence, **to avoid relegation**, a team need **33 points**, on average.

# A Codes

```
import os
import numpy as np
import sys
import scipy
import csv
import time
np.set_printoptions(threshold=sys.maxsize)
num_iter = 1000000

start = time.time()

teams_to_index = ["Arsenal",
"Bournemouth",
"Brighton",
"Burnley",
"Chelsea",
"Crystal Palace",
"Everton",
"Huddersfield",
"Leicester",
"Liverpool",
"Man City",
"Man United",
"Newcastle",
"Southampton",
"Stoke",
"Swansea",
"Tottenham",
"Watford",
"West Brom",
"West Ham"]

# Global var for the params
attack_home = np.zeros(20)
attack_away = np.zeros(20)
defense_away = np.zeros(20)
defense_home = np.zeros(20)
av_home_goals = 0.0
av_away_goals = 0.0

def get_params():
  filename = "season-1718.csv"
  filename2 = "season-1819.csv"

```

```
fields1 = []
fields2 = []
rows = []
rows2 = []

with open(filename, 'r') as csvfile: # 17-18 season
  # creating a csv reader object
  csvreader = csv.reader(csvfile)

  # extracting field names through first row
  fields1 = csvreader.next()

  # extracting each data row one by one
  for row in csvreader:
    rows.append(row)

with open(filename2,'r') as csvfile: # 18-19 season, uptill now
  csvreader = csv.reader(csvfile)

  fields2 = csvreader.next()

  for row in csvreader:
    rows2.append(row)

fields = np.asarray(fields1)
rows = np.asarray(rows)
rows2 = np.asarray(rows2)

fields = fields[1:6]
rows = rows[:,1:6]
rows2 = rows2[:,1:6]
# HomeTeam, AwayTeam, HomeGoals, AwayGoals, Result

tot_games = 0
tot_home_goals = 0.0 # away conceded = home scored
tot_away_goals = 0.0 # home condeded = away scored
tot_games_home_perteam = np.zeros(20) # total number of games each team played home
tot_games_away_perteam = np.zeros(20) # total number of games each team played away
tot_home_perteam = np.zeros(20) # total number of goals each scored at their home
tot_away_perteam = np.zeros(20) # total number of away goals each team scored
tot_conceded_home_perteam = np.zeros(20) # total number of
```

```
    goals team conceded at home
tot_conceded_away_perteam = np.zeros(20) # total number of
    goals team conceded away
global av_home_goals
global av_away_goals

for i in rows:
  tot_home_goals += int(i[2])
  tot_away_goals += int(i[3])
  tot_games += 1

  home_index = teams_to_index.index(i[0])
  away_index = teams_to_index.index(i[1])

  tot_games_home_perteam[home_index] += 1
  tot_games_away_perteam[away_index] += 1

  tot_home_perteam[home_index] += int(i[2])
  tot_away_perteam[away_index] += int(i[3])

  tot_conceded_home_perteam[home_index] += int(i[3])
  tot_conceded_away_perteam[away_index] += int(i[2])

for i in rows2:
  tot_home_goals += int(i[2])
  tot_away_goals += int(i[3])
  tot_games += 1

  home_index = teams_to_index.index(i[0])
  away_index = teams_to_index.index(i[1])

  tot_games_home_perteam[home_index] += 1
  tot_games_away_perteam[away_index] += 1

  tot_home_perteam[home_index] += int(i[2])
  tot_away_perteam[away_index] += int(i[3])

  tot_conceded_home_perteam[home_index] += int(i[3])
  tot_conceded_away_perteam[away_index] += int(i[2])

av_home_goals = float(tot_home_goals/tot_games) # sum of
  all home goals/number of games
av_away_goals = float(tot_away_goals/tot_games) # sum of
  all away goals/number of games


```

```
    for i in range(20):
        attack_home[i] = (tot_home_perteam[i]/
            tot_games_home_perteam[i])/av_home_goals
        attack_away[i] = (tot_away_perteam[i]/
            tot_games_away_perteam[i])/av_away_goals

        defense_away[i] = (tot_conceded_away_perteam[i]/
            tot_games_away_perteam[i])/av_home_goals
        defense_home[i] = (tot_conceded_home_perteam[i]/
            tot_games_home_perteam[i])/av_away_goals

    return
get_params()

# print(av_home_goals)
# print(av_away_goals)
# print(attack_home)
# print(attack_away)
# print(defense_home)
# print(defense_away)
# 0 to 19

# Set up the schedule - form not considered, does not make a
    difference
schedule = np.asarray([[0,1]])
for i in range(20):
    for j in range(20):
#       print(i,j)
        if ((i == (j-1)) and (i == 0)):
            temp = np.asarray([[i,j]])
        elif (i != j) :
            schedule = np.append(schedule,[[i,j]], axis = 0)
np.random.shuffle(schedule)

def league_sim():
    # Set up data structure for the table
    table = np.zeros((20,9)) # ordered as per teams to index
    # Played, Wins, Draws, Losses, Goals For, Goals Against,
        Goal Diff, Points, index
    for i in range(20):
        (table[i])[8] = i

    #print(schedule)

    for match in schedule:
        hometeam_stats = table[match[0]]
```

```
  awayteam_stats = table[match[1]]

  # both play a game
  hometeam_stats[0] = hometeam_stats[0] + 1
  awayteam_stats[0] = awayteam_stats[0] + 1

  # Get the goals both teams scored
  goals_home = np.random.poisson((av_home_goals*attack_home[match[0]]*defense_away[match[1]])) # TODO - how are goals generated
  goals_away = np.random.poisson((av_away_goals*attack_away[match[1]]*defense_home[match[0]])) # TODO - how are goals generated

  if(goals_home > goals_away):
    hometeam_stats[1] += 1 # home won
    awayteam_stats[3] += 1 # away lost

    hometeam_stats[4] += goals_home # GF home
    hometeam_stats[5] += goals_away # GA home

    awayteam_stats[4] += goals_away # GF away
    awayteam_stats[5] += goals_home # GA away

    hometeam_stats[6] += (goals_home - goals_away) # GD home
    awayteam_stats[6] -= (goals_home - goals_away) # GD away

    hometeam_stats[7] += 3 # Winner gets points
  elif(goals_home == goals_away):
    hometeam_stats[2] += 1 # home drew
    awayteam_stats[2] += 1 # away drew

    hometeam_stats[4] += goals_home # GF home
    hometeam_stats[5] += goals_away # GA home

    awayteam_stats[4] += goals_away # GF away
    awayteam_stats[5] += goals_home # GA away

    hometeam_stats[7] += 1 # both get points
    awayteam_stats[7] += 1
  else : #(goals_home < goals_away)
    hometeam_stats[3] += 1 # home lost
    awayteam_stats[1] += 1 # away won

```

```
      hometeam_stats[4] += goals_home # GF home
      hometeam_stats[5] += goals_away # GA home

      awayteam_stats[4] += goals_away # GF away
      awayteam_stats[5] += goals_home # GA away

      hometeam_stats[6] += (goals_home - goals_away) # GD home
      awayteam_stats[6] -= (goals_home - goals_away) # GD away

      awayteam_stats[7] += 3 # Winner gets points

    table[match[0]] = hometeam_stats
    table[match[1]] = awayteam_stats

  #print(table)

  # use mergesort as it is a stable sort
  # the intial sorting remains if later sorting is not able to resolve
  # last by Points, then GD, GF, GA
  table = table[table[:,3].argsort(kind = 'mergesort')[::-1]] # sort by losses, reverse
  table = table[table[:,1].argsort(kind = 'mergesort')] # sort by wins
  table = table[table[:,4].argsort(kind = 'mergesort')] # sort by GF
  table = table[table[:,6].argsort(kind = 'mergesort')] # sort by GD
  table = table[table[:,7].argsort(kind = 'mergesort')[::-1]] # sort by points
  # print(table) # if all above same, alphabetical
  return table

num_wins = np.zeros(20)
av_points = np.zeros(20)

for i in range(num_iter):
  table = league_sim()
  num_wins[int(table[0][8])] += 1
  for j in range(20):
    av_points[j] += table[j][7]

av_points = av_points/num_iter

```

```python
print(num_wins)
print(av_points)

with open("op1.txt","a") as file:
  file.write("new sim starts, num iter is %d\n"% num_iter)
  file.write("num wins is\n")
  for i in range(20):
    file.write("%s - %d \n"% (teams_to_index[i],num_wins[i]))
  file.write(" \n")
  file.write("Average points total is\n")
  for i in range(20):
    file.write("%dth place - %6.2f \n"% (i,av_points[i]))
  file.write(" \n")




end = time.time()
print(end-start)
```