Ranking System

Spring 2020 INFO6205 Project

Team Members

Akshay Phapale, NUID: 001316563

Sanket Pimple, NUID: 001086416

Instructors

Prof. Robin Hillyard,
Aditi Jalkote, Ajay Goel, Hao Cui

Northeastern University
College *of* Engineering

Date: 04/17/2020

Information Systems Department
INFO 6205 34506: Program Structures and Algorithms—Section 03, Spring 2020

Introduction:

The English Premier League being one of the top-level Football (Soccer) leagues in the world, has very interesting and elaborate rules to measure a team's performance and ranking among all its competitors. This makes it a suitable case to research and use for developing a Ranking System where, using historical data, we can calculate the probability $P(x_i, x_j)$ of a team x_i winning against a team x_j. Using the probability values, we can calculate rankings for every team in the season and prepare the final standings.

As the 2019-20 season has ended prematurely due to the COVID-19 pandemic, we don't know how the season would've ended if it were to complete all the matches. Using the probabilistic analysis mentioned above, it is possible to construct a table for 2019-20 season EPL final standings based on calculated predictions. However, there are multiple factors involved in determining this which makes it far from being straightforward. This project tries to solve this exact problem considering various factors involved in an EPL game.

Aim:

To determine the outcome of an EPL game, given two teams, based on probabilistic calculations using the statistics of previous seasons, and ultimately constructing a table of final standings in the season.

Approach:

- For developing the Ranking System, the mathematical concepts of probability and statistics have been used and all the calculations have been made using Java 8 programs.
- The input data is the statistics of every game in all the seasons from 2011 to 2020, in CSV format.
- This data is parsed in the program and the following data is calculated and stored for every team:
    - Number of times Team A played against every other Team B
    - Goal-difference of every match against every other team
    - Probability of a game of Team A vs Team B resulting in a specific goal-difference
    - Mean of goal-difference and Standard Deviations for Team A against every Team B
- These are some of the values calculated in the system. Using these values, an analysis is performed to determine the most probable outcome for a team against any other team
- Using the same data, the final standings for 2019-2020 season are constructed

Input Dataset:

- The input data [1] consists of CSV files for statistics related to every game in each season from 2011 till 2020.
- Every file refers to a specific season
- In each CSV the following fields, among several others have been majorly used:
  - Home Team Name
  - Away Team Name
  - Full Time Home Goals
  - Full Time Away Goals
- Other values like goal-difference, total points etc. are calculated from the above data

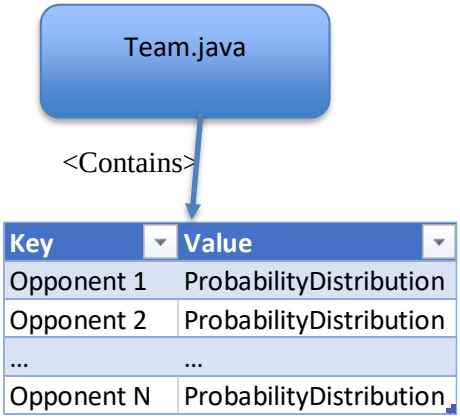
Program:

Data Structures & Classes:

The data structures and classes are described as follows:

- **RankingSystem.java:**
  This is the main class of our project. The main functionality of this class is to read the EPL datasets from season 2011-2012 till season 2019-2020 and to calculate the final standings of season 2019-2020 using our Ranking algorithm.

- **ProbabilityDistribution.java:**
  There is a direct dependency between Teams.java and this class. Refer the below diagram for understanding the relationship:



Data Structure 1 'Map' data

- The single instance of this class represents:
  - Total count of wins, defeats and draws each, of Team A against Team B
  - The Average Goal Difference of Team A against Team B(mean)
  - Two maps which give us how many times Team A has the same goal difference (key-> Goal Difference, Value-> the number of times GD occurred) and the probability against each GD.
- Below are the properties of this class:

```
private Map<Integer, Double> probabilityOfEvent; //key = GD, value =
probability of getting that GD among all
// matches played
private Map<Integer, Integer> noOfTimesEventOccurred; //key = GD, value =
count of matches resulting in this GD
private int count; //count of matches played
private double mean; //mean of GD of all matches played against this team,
i.e. Team A vs Team B
private double winCount, loseCount, drawCount;
```

E.g. Let us see the probability distribution instance for Liverpool against Fulham.

Two more important methods which belong to this class are

- getMean()
- getSD()

These two methods will calculate the average goal difference of Team A against Team B and average standard deviation in goal difference.

- **Team.java**

This class represents the data associated with a particular team. The single instance of this class represents the name of the team(name), the score of the team(score), and a map that has Key as opponent team name and values as Probability Distribution of Team against the opponent(teamstats).

Below are the properties of this class:

```
private String name;
private Map<String, ProbabilityDistribution> teamStats; //key = opponent,
value = probability distribution of
// goals against that team
private int score;
```

Two more important methods belong to this class and those methods are getAvgMean() and getAvgSD(), these two methods will calculate the average goal difference of this team and average standard deviation in goal difference.

- **Teams.java**

  - This class represents the collection of Team objects. This class acts as an abstraction to create, manage and supply Team instances and related data.
  - This class contains one more method i.e. compareTeams() (See Below) that calls the TeamComparator class methods (static).

```java
/**
 * This method compares the two teams by using the probability data stored
in the respective Team instances of
 * the given teams, forwards the objects to a detailed comparator method
which checks mean GD and Standard
 * deviation of the mean GD and probabilities to return a predicted GD
value
 * @param teamNameA name of team A
 * @param teamNameB name of team B
 * @return Approximate predicted goal difference between two given teams
 */
public Integer compareTeams(String teamNameA, String teamNameB) {
    Team teamA = this.lstTeam.stream().filter(x ->
x.getName().equalsIgnoreCase(teamNameA)).findFirst().orElse(null);
    Team teamB = this.lstTeam.stream().filter(x ->
x.getName().equalsIgnoreCase(teamNameB)).findFirst().orElse(null);
    if (teamA != null && teamB != null) {
        return TeamComparator.compare(teamA, teamB); // Team A win or
loose by what margin
    }
    return null;
}
```

- **TeamComparator.java**
    - This class represents our algorithm to rank or compare two teams based on various parameters. We will discuss this class in more detail in the below section.

Algorithms and Formulae:

Team Ranking algorithm:

Step1: Get the Team instances of two teams those need to be compared.

Step2: Get the Probability Distribution instances of both teams.

Step3: Using Probability Distribution check if two teams have previous records (More than 4 matches) against each other else Go to Step 5

Step4: Return Goal Difference using getMean() method from the probability distribution instance.

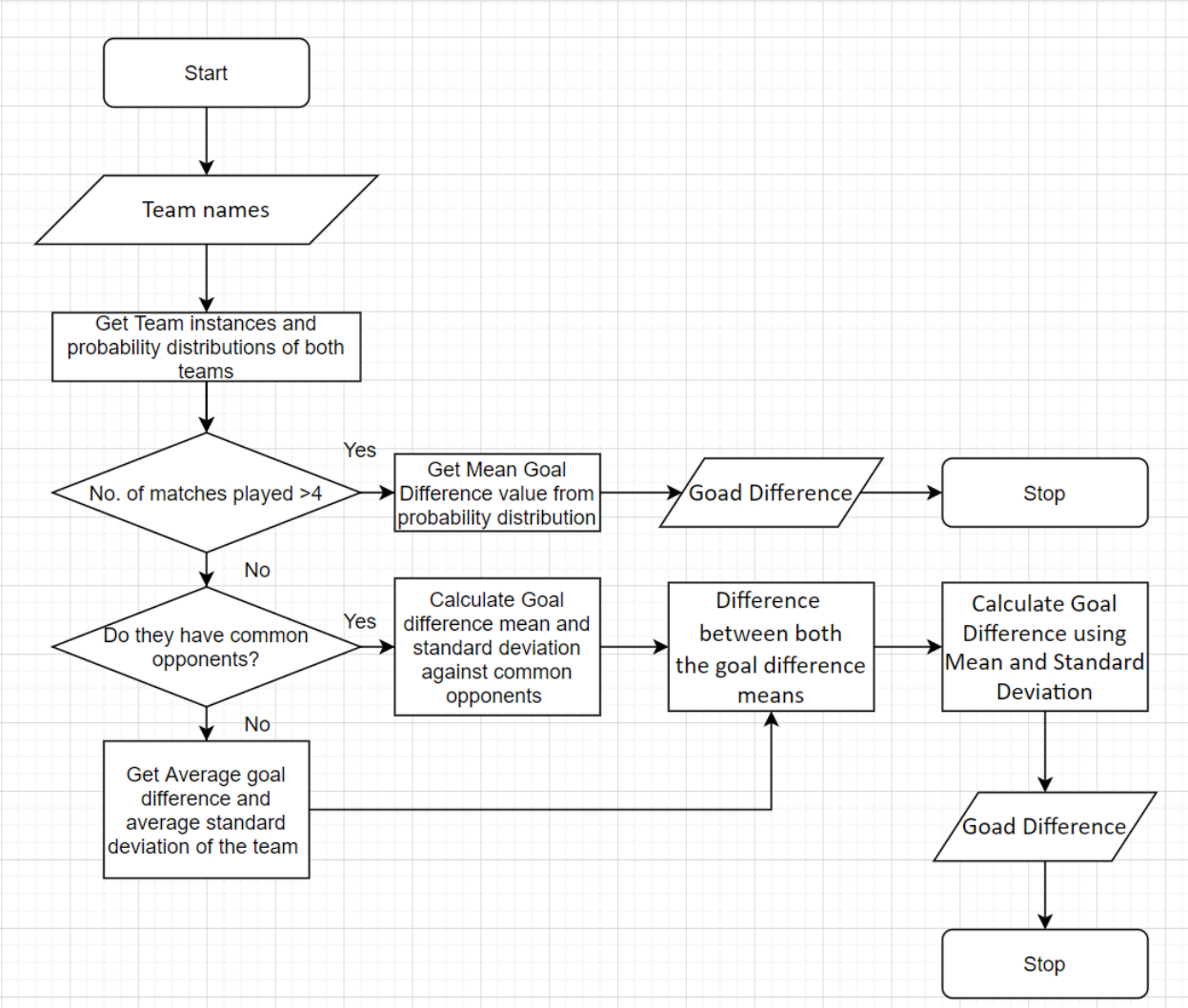
Step5: Check if both the team shares common opponents else Go to step 7

Step6: Calculate the Average Mean and Average Standard Deviation of goal difference of both teams against common opponents and then Return the Difference between both means as a result.

Step7: Get the Average Mean and Average Standard Deviation of both the teams and return the Difference between both means as a result.

***Note**: Whenever the Means are very close to each other (normal distribution graphs almost overlap each other) then to differentiate between them we can consider the standard deviation for result calculation. The team with less standard deviation is considered to be more consistent, so we will be giving results by comparing standard deviation values in such cases.*

Ranking System Flowchart:



Formulae:

To calculate the mean, we use a general formula that first calculates the sum and then adds the new value to the sum, and divides the new sum by the new count.

```
this.mean = (this.mean*(count-1)+goalDifference)/count;
```

To calculate the standard deviation of the sample:

The **Sample** Standard Deviation: $s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2}$

```
public double getSD(double mean){
    double difference = 0;
    for (Map.Entry<Integer, Integer> entry :
this.noOfTimesEventOccurred.entrySet()) {
        difference += Math.pow(entry.getKey()-mean,2)*entry.getValue();
    }
    difference = difference/(count-1);
    return Math.sqrt(difference);
}
```

To merge two means: Calculating the sum of both the means and then calculating the new mean.

```
public double getAvgMean(List<ProbabilityDistribution> teamsPD) {
    double mean = 0;
    int count = 0;
    for (ProbabilityDistribution pd : teamsPD) {
        double mean2 = pd.getMean();
        int count2 = pd.getCount();

        mean = ((mean * count) + (mean2 * count2)) / (count + count2);
        count = count + count2;
    }
    return mean;
}
```

To merge two standard deviations:

$$S_b' = \sqrt{\frac{(n_1-1)S_1^2+(n_2-1)S_2^2}{n_1+n_2-2}}$$

```
public double getAvgSD(List<ProbabilityDistribution> teamsPD) {
    if (teamsPD.size() == 1) {
        ProbabilityDistribution pd = teamsPD.get(0);
        return pd.getSD(pd.getMean());
    }
    double sd = 0;
    int count = 0;
    for (ProbabilityDistribution pd: teamsPD) {
        double sd2= pd.getSD(pd.getMean());
        int count2= pd.getCount();
        sd = Math.sqrt(((count)*Math.pow(sd,2) +
(count2)*Math.pow(pd.getSD(pd.getMean()),2))/(count+count2));
        count = count+count2;
    }
    return sd;
}
```

Results and Analysis:

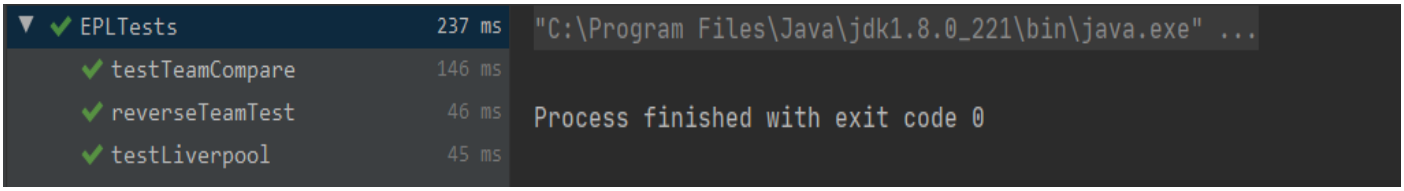
Output:

- Calculated standings at the end of the season 2019-2020

Rank	Team Name	Goal Difference	Points
1	Liverpool	52	103
2	Man City	54	85
3	Chelsea	18	68
4	Man United	20	64
5	Leicester	29	63
6	Tottenham	13	60
7	Sheffield United	7	58
8	Arsenal	6	54
9	Wolves	6	51
10	Burnley	-7	49
11	Crystal Palace	-8	47
12	Southampton	-16	46
13	Everton	-10	46
14	Newcastle	-22	39
15	West Ham	-16	35
16	Brighton	-15	35
17	Bournemouth	-24	33
18	Watford	-24	32
19	Aston Villa	-31	29
20	Norwich	-32	27

(output of result.csv from GitHub)

- The output of test cases:



- Command-line user input result:

```
------------------------Get probability of a team winning or losing:-------------------------
Please enter the full names of 2 teams of your choice, comma separated :
Arsenal, Chelsea
Arsenal has following probabilities of full-time result against Chelsea:
Winning:0.16666666666666666
Losing:0.5
Draw:0.3333333333333333
Arsenal is most likely to lose against Chelsea with an average goal-difference of approximately 1

Process finished with exit code 0
```

Observations:

- Liverpool vs Arsenal goal difference graph:

From the above graph, we can see that Liverpool has a high chance of winning against Arsenal as number of wins > number of losses.

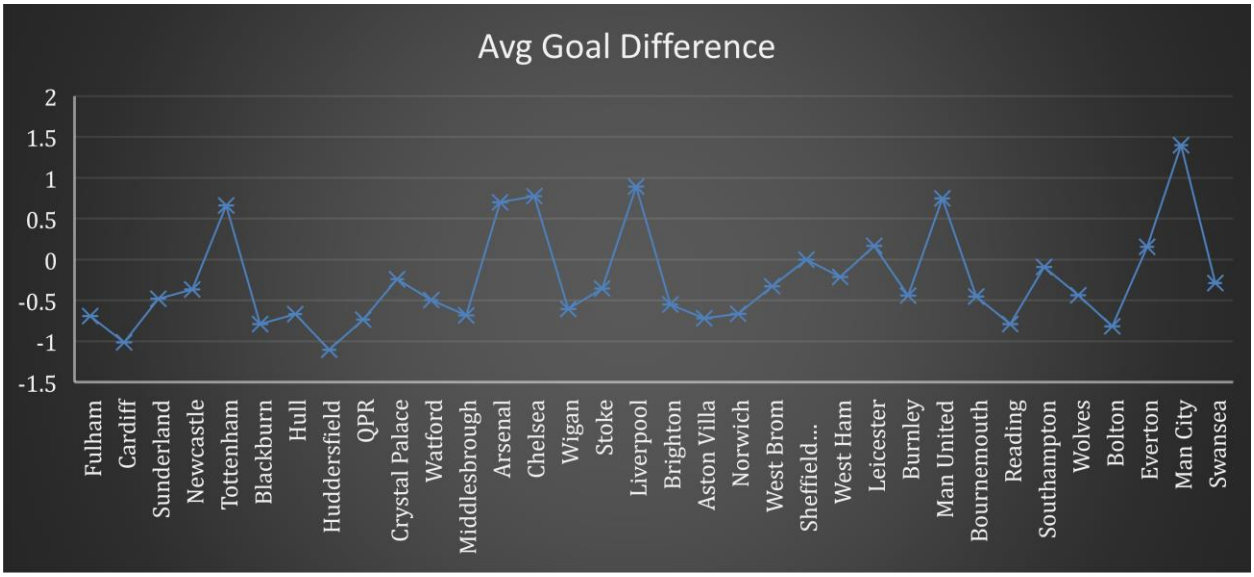
- Let's test this against command line input.

```
Please enter the full names of 2 teams of your choice, comma separated :
Liverpool, arsenal
Liverpool has following probabilities of full-time result against arsenal:
Winning:0.4117647058823529
Losing:0.23529411764705882
Draw:0.35294117647058826
Liverpool is most likely to win against arsenal with an average goal-difference of approximately 1
```

Here mean of Liverpool against Arsenal is approximately **0.64**

```
mean = 0.6470588235294118
```

- Let's see the graphical representation of the average goal difference of all the teams:



From the graph it is clear that Man City, on an average, scores more goals which perfectly matches with the current circumstances of Manchester City since 2011.

Edge cases:

There are some edge cases which have been considered while calculating all the values related to every team:

- Simulation of a game between two teams which have never played against each other:
    - Although this case might not seem to be very likely if we reply on our intuitions, it cannot be ignored.
    - As per the algorithm above, if such situation occurs, the code will first look for data related to **common opponents** these teams have played against. If found, the code will calculate how these teams **perform against those common opponents**. The **average mean Goal-difference** and the **standard deviation** are calculated and compared to determine the team which has the highest probability of winning.
    - If there are no common opponents, the data related to the team's overall performance in the past is considered
- In many situations the average probable goal-difference results to 0 because of rounding-off of values. Being cognizant of this situation, the code also considers other probability values like win/loss/draw probability before drawing a conclusion.

Extra Analysis:

According to a research mentioned in this article [4] - (https://www.inverse.com/article/61836-premier-league-partial-standings), which states,

"after the 10th game of a 38-game Premier League season, there's a 77 percent chance one's team is going to finish in or around its position in the table (standings). There's no argument, it's just the math"

We can see such similarities between the current standings of an incomplete season and the final table calculated (as shown above) by our Ranking System, especially in the first four and the last four teams in the table.

More Factors that can be considered which will improve the accuracy of the system (Future Implementations):

As we know that the EPL consists of numerous interdependent factors which define the games and their outcomes, resulting in series of complex events with an enormous scope of analysis. Following are some more factors that can be derived from or are related to the calculations done in this Ranking System, which will improve the accuracy of the system and provide more insight:

1. Calculating top five performing players in every team and checking how many of them played in each match, which gives rise to these cases:

    - If fewer of them played and the team won, then P(win) should increase
    - If all played and the team lost or ended up with a draw, then P(win) should decrease
2. Calculating the probability of winning at home & away
3. Checking correlation between the number of recent transfers and the game after the transfers
4. Probability of an opposite score (comeback/downfall) in full time as compared to halftime

Conclusion:

The Ranking System based on concepts of probability and statistics has helped in gaining insights related to,

- possible outcomes of a game between two teams in the EPL.
- how these values affect the final standings of a season
- what are different factors involved in deciding the rank of a team at the end of a season
- how we can improve the System to perform more detailed analysis and increase the accuracy of our results

References:

[1] Input Data http://www.football-data.co.uk/englandm.php

[2] Calculating the Standard Deviation of a sample

https://www.mathsisfun.com/data/standard-deviation-formulas.html

[3] Combining Standard Deviation of two groups

https://math.stackexchange.com/questions/2971315/how-do-i-combine-standard-deviations-of-two-groups/2971522#2971522

[4] Extra Analysis https://www.inverse.com/article/61836-premier-league-partial-standings

[5] The Premier League https://www.premierleague.com/