Distributed FC

December 8, 2022

Walter
Ryan
Joshu
CSC 369-03

Description:

Our project takes data from various soccer teams and matches to make projections as to which team will win, and attributes that tend to give teams more success. Using Spark/Scala structures and operations, along with some linear regression modeling, we believe to have produced enough quantifiable results to make predictions for successful teams, and which team will more likely win a matchup

Goals:

1. Predict which team will win in a given matchup
2. Compare home vs away advantages/disadvantages
3. Predict team success given a set of team attributes

Glossary:

xG - expected goals
xGA - expected goals against
BU - build up (a team's build up play category stat)
CC - chance creation (a team's chance creation category stat)
D - defense (a team's defense category stat)
GD - goal difference
LM - linear regression

Database and Tables used:

European Soccer Database
https://www.kaggle.com/datasets/hugomathien/soccer?resource=download

All input files are cleaned and sorted from the Kaggle database using SQL

Teams

`Id`: Int

`name`: String

Team_Attributes

`Id`: Int

`date`: Date()

`buSpeed`: String options: slow, balanced, fast

`buDribbling`: String options: little, normal, lots

`buPassing`: String options: short, mixed, long

`ccPassing`: String options: little, normal, lots

`ccCrossing`: String options: little, normal, lots

`ccShooting`: String options: little, normal, lots

`ccPositioning`: String options: free form, organized

`dPressure`: String options: options: medium, deep, high

`dAggression`: String options: press, double, contain

`dWidth`: String options: normal, narrow, wide

`dLine`: String cover, offsides trap

Matches

`Id`: Int

`homeTeamID`: Int

`awayTeamID`: Int

`homeGoals`: Int

`awayGoals`: Int

1. Predicting the winner in a given matchup

The predictions work by comparing the way the 2 teams would perform in their next matchups based on their recent form. After building a linear regression model with our TrainLR.csv that contains data on all the teams. We then store the data on the 2 teams we want to predict their next matchup in our TestLR.csv. The outcomes are determined to mean that the team with the higher goal difference prediction has a higher chance of winning their next game. In the event that the predictions are the same then we can conclude that both teams would draw. We then verify our results by collecting the actual score of both our teams next game to see if the winner's we predicted are correct. In the event we wanted to account for more things like a home team advantage we would have to add a new feature that adjusts our predictions for a single team in our TestLR.csv

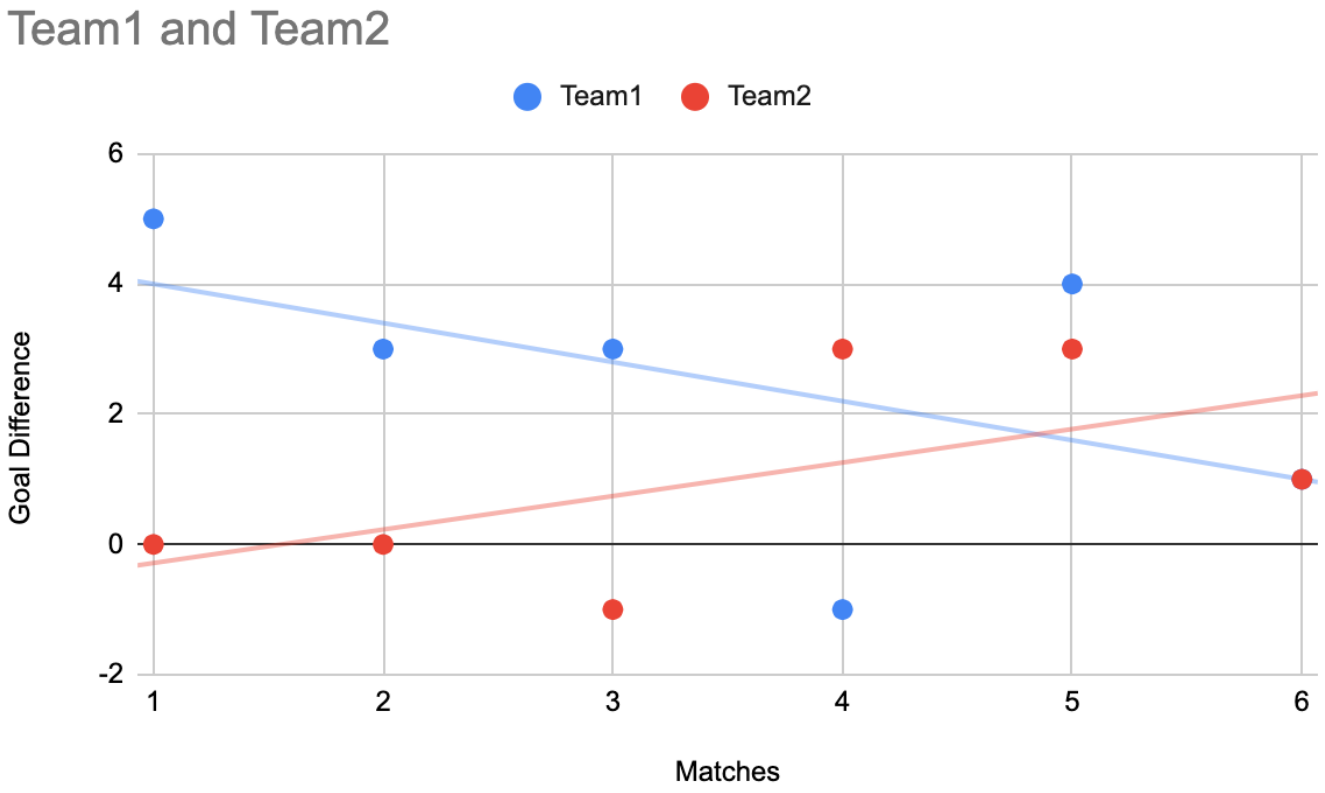
```
+----+-----+------------------+
|TEAM|label|         prediction|
+----+-----+------------------+
|8668| -1.0|-0.6642992160532588|
|9825|  2.0| 1.9398845084167744|
+----+-----+------------------+

Coefficients: [-0.28866994326675727,-0.09073237303776836,-0.11200708336644062,0.4704609020972457,0.18834584702446838,2.8341133758428296] Intercept: -0.006873615977057705
Team ID: 8668 Actual score: 0
Team ID: 9825 Actual score: 1
```

Data:

Matches	Team1	Team2
1	5	0
2	3	0
3	3	-1
4	-1	3
5	4	3
6	1	1

Data model:



2. Comparing home/away advantage

Historically, it is thought that the home team holds an advantage over the away team. While this belief makes sense (in our opinion at least), we put the data together to prove if this was true or not.

Method:

- Join all matches to respective teams
- Calculate record and split by home or away
- Group by home record and away record
- Aggregate records

Code

```
// teams with records split home/away
// ((Team(), home/away), Map(numLoss -> Int, numWin -> Int, numDraw -> Int)
// home = 1, away = -1
// loss = -1, win = 1, draw = 0
val teamWithRecordHA = sc.parallelize(matchData.map { x => {
  List((x.homeID, x), (x.awayID, x))
}}.collect().toList.flatten).rightOuterJoin(teamData.map { x => {
  (x.id, x : Team)
}}).map { case (_, (m, team)) => {
  (team, m.getOrElse(None))
}}.filter(_._2 != None).map{case (t, m) => {
  var currMatch = m.asInstanceOf[Matches]
  ((t, findHomeAway(currMatch, t.id)), calculateMatchOutcome(currMatch, t.id))
}}.groupByKey().mapValues(x => x.groupBy(identity).mapValues(_.size))
```

```
// finds w/d/l % of home and away teams
// ("home" or "away", (win%, draw%, loss%))
val aggregatedRecordsOfHA = teamWithRecordHA.map({case ((_, ha), record) => {
  (ha, record)
}}).reduceByKey(combineRecords(_, _)).map{case (ha, record) => {
  var totalGames = record.foldLeft(0)(_+_._2).toDouble
  (if(ha == 1) "home" else "away",
    (record.get(1).get.toDouble / totalGames, record.get(0).get.toDouble / totalGames, record.get(-1).get.toDouble / totalGames))
}}
```

Output:
([“home”, “away”], (win, draw, loss)) *record as percentage of total games played

```
(away,(0.28738596558759,0.25389737865198814,0.45871665576042187))
(home,(0.45871665576042187,0.25389737865198814,0.28738596558759))
```

Result:

Home teams win about 45.9% and lose about 28.7% with 25.4% of match results ending in a draw. This seems to align with findings from reference 1:

> In football leagues across Europe, for example, prior research has found that home teams win an average of 50% of matches played in their stadiums.
>
> Variations exist, of course. In Germany’s Bundesliga, football teams playing at home only win an average of 47% of games at their home stadiums.

3. Predicting team success based on attributes

The database we found was gracious enough to include sets of attributes for each team. These attributes include:

- Build up (bu)
    - Speed
    - Passing
    - Dribbling
    - Positioning
- Chance creation (cc)
    - Passing
    - Crossing
    - Shooting
    - Positioning
- Defense (d)
    - Line
    - Width
    - Aggression
    - Pressure

The main metric used to calculate a good score for bu and cc stats is expected goals (xG). For measuring d attributes, we used expected goals against (xGA). In this case, the value is reversed—the lower the xGA the better. Here we hope to see which combinations of attributes gave a team the highest projected xG, and lowest projected xGA.

Method:

- S1: Join teams on matches and calculate xG
- S2: Join teams on matches and calculate xGA
- S3: Join S1 on team attributes
- S4: Join S2 on team attributes
- Group S3 by bu attributes
    - Sort by xG DESC
- Group S3 by cc attributes
    - Sort by xG DESC
- Group S4 by d attributes
    - Sort by xGA ASC

Code:

```
// teams w associated team attributes
// (TeamID, TeamAttributes())
// (9825,TeamAttributes(9825,Balanced,Normal,Short,Organised,Safe,Normal,Normal,Free Form,Medium,Press,Normal,Cover))
val teamWithAttributes = teamData.map(x => (x.id, x.name)).leftOuterJoin(teamAttributeData.map(x => (x.id, x : TeamAttributes)))
  .map{case (id, (name, attr)) => {
    (id, attr.getOrElse(None))
  }}.filter(_._2 != None);
```

```
// find xG for team
// (Team, xG)
val teamWithxGFor = sc.parallelize(matchData.map { x => {
  List((x.homeID, x), (x.awayID, x))
}}.collect().toList.flatten).rightOuterJoin(teamData.map { x => {
  (x.id, x : Team)
}}).map { case (_, (m, team)) => {
  (team, m.getOrElse(None))
}}.map{ case (t, m) => {
  (t, findNumGoalsFor(t.id, m.asInstanceOf[Matches]))
}}.groupByKey().map{case (t, xG) => (t.id, (t.name, xG.sum.toDouble / xG.size.toDouble))}
```

```
// find xGA for team
// (Team, xGA)
val teamWithxGAgainst = sc.parallelize(matchData.map { x => {
  List((x.homeID, x), (x.awayID, x))
}}.collect().toList.flatten).rightOuterJoin(teamData.map { x => {
  (x.id, x : Team)
}}).map { case (_, (m, team)) => {
  (team, m.getOrElse(None))
}}.map { case (t, m) => {
  (t, findNumGoalsAgainst(t.id, m.asInstanceOf[Matches]))
}}.groupByKey().map { case (t, xG) => (t.id, (t.name, xG.sum.toDouble / xG.size.toDouble)) }
```

```
// determine which combination of build up play attributes has the highest xG
// ((buSpeed, buPassing, buDribbline, buPositioning), xG)
/*
  top 3 results:
  ((Balanced,Short,Little,Free Form),2.400735294117647)
  ((Balanced,Short,Normal,Free Form),1.993421052631579)
  ((Balanced,Mixed,Normal,Free Form),1.739363113666519)
 */
val xGFromBUAttributes = teamWithxGFor.rightOuterJoin(teamWithAttributes).map { case (_, (xG, attr)) => {
  var tAttr = attr.asInstanceOf[TeamAttributes]
  ((tAttr.buSpeed, tAttr.buPassing, tAttr.buDribbling, tAttr.buPositioning), xG.get._2)
}}.groupByKey().mapValues(x => x.sum / x.size).collect().sortBy(_._2 * -1)
```

```
// determine which combination of chance creation attributes has the highest xG
// ((ccPassing, ccCrossing, ccShooting, ccPositioning), xG)
/*
  top 3 results:
  ((Little,Little,Little,Free Form),2.400735294117647)
  ((Little,Normal,Little,Free Form),1.9184769521843827)
  ((Normal,Normal,Normal,Free Form),1.5470414357924358)
 */
val xGFromCCAttributes = teamWithxGFor.rightOuterJoin(teamWithAttributes).map{case (_, (xG, attr)) => {
  var tAttr = attr.asInstanceOf[TeamAttributes]
  ((tAttr.ccShooting, tAttr.ccCrossing, tAttr.ccShooting, tAttr.ccPositioning), xG.get._2)
}}.groupByKey().mapValues(x => x.sum / x.size).collect().sortBy(_._2 * -1)
```

```
// determine which combination of defense attributes has the lowest xGA
// ((dLine, dWidth, dPressure, dAggresion), xGA)
/*
  top 3 results:
  ((Cover,Normal,High,Press),0.7757352941176471)
  ((Cover,Normal,Medium,Double),1.1231617647058822)
  ((Offside Trap,Normal,Medium,Press),1.3288201178156809)
 */
val xGFromDAttributes = teamWithxGAgainst.rightOuterJoin(teamWithAttributes).map { case (_, (xGA, attr)) => {
  var tAttr = attr.asInstanceOf[TeamAttributes]
  ((tAttr.dLine, tAttr.dWidth, tAttr.dPressure, tAttr.dAggresion), xGA.get._2)
}
}.groupByKey().mapValues(x => x.sum / x.size).collect().sortBy(_._2)
```

Results:

When analyzing build up play attributes, teams that employed a balanced build up speed and free form positioning performed the best, averaging the highest xG per match. The code for matching those attributes to corresponding teams is pretty trivial, so I won't show it here. However, the results are in line with expected. These teams included historically strong teams like Manchester City, Barcelona and Real Madrid.

Teams that matched the highest xG per chance creation attributes contained mixed results, but Barcelona, Villareal, and Madrid were among the top teams.

For defensive stats, teams that matched with the attributes corresponding to the lowest xGA included Bayern, Dortmund and leverkusen— the top three teams in the German Bundesliga that season.

References

1. https://www.courthousenews.com/study-finds-home-field-advantage-in-soccer-doesnt-rely-on-spectators/