Doing a basic team ranking using match data

October 22, 2013

This shows how to use a comma-delimited file of match data to get a team ranking and then to predict tournaments. The example uses match and team data downloaded from www.lastplanetranking.blogspot.com.

1 Load the data

The ranking function needs a dataframe. To get one, we will use the `create.fbRanks.dataframes()` function which reads in a comma-delimited match file, does error checking and creates a dataframe. But you can create a dataframe any way you want.

We will start by loading the example .csv file into the current workspace.

```
pkgpath=find.package("fbRanks")
file.loc=paste(pkgpath,"\\doc\\scores-web.csv",sep="")
file.copy(file.loc,".")
```

```
temp=create.fbRanks.dataframes(scores.file="scores-web.csv")
```

```
Alert: teams info file was not passed in.
Will construct one from the scores data frame but teams in the scores file must use a unique name.
Alert: teams resolver was not passed in.
Will construct one from the team info data frame.
```

```
scores=temp$scores
```

In the example scores file, each team has a unique name thus a ”team resolver” file is not needed. A ”team resolver” file is needed when our match data has one team using different names, which will happen if you have data from different tournaments and leagues. The example scores file already has the names standardized for you. This shows the first few lines of the example match dataframe. The example scores dataframe also has venue information and surface information (try `head(scores)`) but only the columns shown below are necessary for a basic ranking.

```
head(scores[,1:5])
```

```
        date                home.team home.score                away.team away.score
1 2012-05-25          PacNW Blue B00          0 Normandy Park FC White B00          8
2 2012-05-25        WA Rush Swoosh B00          1          PacNW Maroon B00          7
3 2012-05-26    Crossfire Premier B B00         12        WA Rush Swoosh B00          0
4 2012-05-26 Normandy Park FC White B00          4          WA Rush Nike B00          0
5 2012-05-26           PacNW Maroon B00          1    Crossfire Premier B B00          8
6 2012-05-26          WA Rush Nike B00         10            PacNW Blue B00          1
```

2 Rank the teams

To do just a basic ranking, the `rank.teams()` function requires a dataframe with date, home.team, home.score, away.team, and away.score columns that are respectively class date, character, numeric, character, and numeric.

```
#get the ranks without any explantory variables
ranks1=rank.teams(scores=scores)
```

We can print the ranks. The output is not shown since there are 200+ teams.

```
#print all ranks
print(ranks1)
```

The problem with not using a team file is that we have no information on the teams, like the region or the age. Let's add a team file which has the region for each team. Again this is a file downloaded from www.lastplanetranking.blogspot.com. The file has a variety of information about the teams.

```
temp=create.fbRanks.dataframes(scores.file="scores-web.csv", teams.file="teams-web.csv")
```

```
Alert: teams resolver was not passed in.
Will construct one from the team info data frame.
```

```
scores=temp$scores
teams=temp$teams
head(teams[,c("name","age","region","fall.league")])
```

```
                                   name   age region  fall.league
1                       Coastal FC 2000B 2000B     BC BCSPL U13fin
2 Coquitlam Metro Ford Millennium 2000B 2000B     BC BCSPL U13fin
3                        Fusion FC 2000B 2000B     BC BCSPL U13fin
4             KSYSA Warriors Fogal 2001B 2001B     BC
5       Magnuson Ford Mariners FC 2000B 2000B     BC BCSPL U13fin
6                 Mountain United 2000B 2000B     BC BCSPL U13fin
```

We can fit again, this time with a team resolver file:

```
#get the ranks without any explantory variables
ranks2=rank.teams(scores=scores, teams=teams)
```

Now we can print using the columns in the teams dataframe. Show the ranks for teams in the fall RCL Division 1 U12 league.

```
 print(ranks2, fall.league="RCL D1 U12")
```

```
Team Rankings based on matches 1900-05-01 to 2100-06-01
fall.league: RCL D1 U12
  team                    total attack defense n.games fall.league
1 WPFC Black B00          6.70  7.72   13.48   32      RCL D1 U12
2 Eastside FC Red B00     6.18  7.59    9.55   29      RCL D1 U12
3 Seattle United Copa B00 6.15  9.36    7.58   20      RCL D1 U12
4 FC Alliance A B00       5.01  7.29    4.41   31      RCL D1 U12
5 Crossfire Premier B B00 4.79  7.94    3.48   40      RCL D1 U12
6 SSC Shadow A B00        4.20  6.32    2.90   25      RCL D1 U12
7 WA Rush Nike B00        4.08  5.28    3.21   34      RCL D1 U12
8 Crossfire Premier A B01 3.94  5.98    2.56   22      RCL D1 U12
```

By default it is scaling the total column by the median total for all teams in the database (not just in RCL D1 U12). What you scale to does not really matter. It is the difference in total strength (difference in the values in the total columns of two teams) that tells you the relative strength. The meaning of the total column is that 2^{diff}, where diff=(total strength team A - total strength team B), is the expected number of goals by team A divided the expect number of goals by team B in a match up between these two teams. So if the difference in total strength is 1 goal, then team A scores twice as many goals as team B, so for every 2 goals scored by team A, team B scores (on average) 1 when team A plays team B. If the difference is 2, team A scores $2^2 = 4$ times as many goals. The scaling constant c, which is (total + c), falls out when you take the difference in total strength between teams so only matters from a presentation perspective.

3 Ranking with explanatory variables

If we want to include explanatory variables in our model, then our scores dataframe also needs columns for the game-specific explanatory variables. In this example, I added surface and 'advantage' to the scores dataframe. Surface specifies whether the game was played on turf, grass or unknown and advantage specifies home, away or neutral advantage.

```
names(scores)
```

```
[1] "date"       "home.team"  "home.score" "away.team"  "away.score" "venue"      "home.adv"
[8] "away.adv"   "surface"
```

The explanatory variables are specified in the `rank.teams` call using `add=`. To include the surface information in our model, add `add="surface"` to the `rank.teams` call.

```
#get the ranks with surface effect; note surface must be a column
#in scores (or teams) dataframe for this to work
ranks3=rank.teams(scores=scores,teams=teams,add="surface")
```

Note, the explanatory variables should be specified as character or numeric values and not as factors.

Explanatory variables that affect the home team score differently than the away team score must be specified with two columns in the scores dataframe. One is called ”home.xyz” and the other is ”away.xyz”. For example, if you wanted to look at the effect of the goalkeeper jersey color on scores (which interestingly has been shown to affect the goals scored against), you could have a column called ”home.GKcolor” and ”away.GKcolor”. Or if you wanted to look at whether the distance teams much travel affects the scores, you could have a column called ”home.traveldist” and ”away.traveldist”. However, the most common home/away explanatory variable is simply whether the team is home or away. In the scores dataframe, I use columns ”home.adv” and ”away.adv” to specify the advantage the team has, which will be either ”home”, ”away” or ”neutral”.

To add an explanatory variable that affects the home and away teams differently, add only the part after ”home.” and ”away.” to the `add=` vector in the `rank.teams` call. Here’s how to add surface and advantage to the B00 ranks:

```
ranks4=rank.teams(scores=scores,teams=teams,add=c("surface","adv"))
```

Slightly fewer goals per game are scored on turf,

```
coef(ranks4$fit$cluster.1)["surface.fTurf"]
```

```
surface.fTurf
   -0.2151422
```

and the home team has an advantage, in this case small.

```
coef(ranks4$fit$cluster.1)["adv.fhome"]
```

```
adv.fhome
0.1215774
```

What is the `cluster.1` bit? Clusters are the groups of teams that are completely connected (there is a path from every team to every other team). You can have multiple clusters in your dataset and `rank.teams` will rank each separately but you will not be able to compare ranks across clusters. The fits are named cluster.1, cluster.2, etc. In this example, we have only one cluster since all the teams in this scores dataframe are interconnected.

4 Predicting and simulating matches

We can easily predict matches and tournaments. First let's fit a model using just the summer data.

```
#Drop the home/away advantage since all the summer games are neutral
#Add max.date so tell the function to only use matches up to max.date
ranks.summer=rank.teams(scores=scores,teams=teams,add=c("surface"), max.date="2012-9-5")
```

Now we can use that model to simulate the outcome of the RCL D1 fall league:

```
simulate(ranks.summer, venue="RCL D1")
```

```
                        1st 2nd 3rd 4th 5th 6th 7th 8th
FC Alliance A B00         0   0   1   6  30  32  21  10
Crossfire Premier B B00   3  10  23  48  11   3   1   0
```

```
WPFC Black B00           22  39  27  11   1   0   0   0
Eastside FC Red B00      11  27  38  21   3   1   0   0
Seattle United Copa B00  64  24   9   2   0   0   0   0
WA Rush Nike B00          0   0   0   2  14  25  35  24
SSC Shadow A B00          0   0   0   0   4  11  25  59
Crossfire Premier A B01   0   0   2   9  36  29  18   7
```

We can also predict the match outcomes for RCL D1 games on the 16th of September. Note the format for a date and that it is wrapped in `as.Date()`.

```
 predict(ranks.summer, venue="RCL D1", date=as.Date("2012-09-16"))
```

```
Predicted Match Results for 1900-05-01 to 2100-06-01
Model based on data from 1900-05-01 to 2012-09-05
---------------------------------------------
2012-09-16 Crossfire Premier B B00 vs Crossfire Premier A B01, HW 61%, AW 21%, T 18%, pred score 2.6-1.5  actu
2012-09-16 WPFC Black B00 vs WA Rush Nike B00, HW 82%, AW 5%, T 13%, pred score 2.6-0.5  actual: HW (3-0)
```

We can construct "fantasy" tournaments by making a dataframe with NaN for the score. The columns date, home.team, home.score, away.team, away.score must be present. The date column must be there but will be ignored. Also, any explanatory variables must be present.

Let's have a fantasy tournament of the three all-city teams in the Seattle United Club and the top Seattle United regional team.

```
 fantasy.teams=c("Seattle United Copa B00","Seattle United Tango B00",
                 "Seattle United Samba B00","Seattle United S Black B00")
 home.team=combn(fantasy.teams,2)[1,]
 away.team=combn(fantasy.teams,2)[2,]
 fantasy.games=data.frame(
   date="2013-1-1",
   home.team=home.team,
   home.score=NaN,
   away.team=away.team,
   away.score=NaN, surface="Grass",
   home.adv="neutral", away.adv="neutral")
```

`fantasy.games` is now a dataframe with an imaginary bracket with 4 teams and each team playing 3 games. Our imaginary tournament is on grass.

```
simulate(ranks4, newdata=fantasy.games, points.rule="tournament10pt")
```

```
                          1st 2nd 3rd 4th
Seattle United Copa B00     95   5   0   0
Seattle United Tango B00     5  86   9   1
Seattle United Samba B00     0   7  58  35
Seattle United S Black B00   0   3  33  64
```

5 Multiple clusters of teams

If you do not have a path from every team to every other team in your database, then you will have some groups of teams that are in different 'clusters'. Let's say you have six teams. A played B, B played C, C played D, and E played F. There is a 'path' from A to D (through the B-C and C-D games) but no path from E or F to A, B, C or D. So A,B,C,D is one 'cluster' and E,F is another. We can rank all the teams within a cluster to each other, but we cannot rank one cluster against another. The fbRanks package will recognize the different clusters and print the ranks separately. However, the `predict` and `simulate` functions expect fbRanks objects (the output from a `rank.teams` call) with only one cluster.