Webscraping Match Data: example using B00s in WA, OR, and BC

October 22, 2013

Collecting the match data is a time-consuming task. The fbRanks package has a number of webscrapers that were built to webscrape US West Coast league and tournament data. Html is fluid and webscrapers quickly succumb to code rot. However, the webscraping code will give you a start for writing your own webscrapers. My webscrapers use the XML, RCurl, httr, and RJSONIO packages. There is some overlap between what these packages do, but I found that some pages could be scraped easily with functions in one package while another page was more easily scraped with functions from another package.

The scrapers are written for particular content manager platforms that are used to show schedules and results. On the US west coast, Demosphere, SportAffinity, GotSport and Korrio are the main platforms. Demosphere and Korrio will also supply surface information (Turf/Grass) that can be scraped. Other leagues use idiosyncratic webpages for showing match results and these require custom scrapers. Finally, some leagues heavily use Javascript in displaying match results, which makes is very hard to scrape unless you can decipher a link to the JSON data.

Adding explanatory variables to your match data

The required information for a match is the date, home team, home score, away team and away score. These must have the column names `date`, `home.team`, `home.score`, `away.team` and `away.score` in the files output by the scrapers.

Often you will want to add other information about a match because you want to be able to filter later on that information (e.g. you want to only show teams that played in a certain tournament) or you want to use that information in your model. For the latter, note that there are two scores for a match, the home score and the away score. Some information might affect both teams regardless of whether they are home or away. For example, the surface (Turf/Grass) and format (9v9, 11v11) affect both teams. Other information is home or away team specific. For example, home advantage is team specific. The home team has advantage ”home” while the away has advantage ”away” (or ”none” or whatever you want to call it). On neutral territory, the home and away teams would have both ”neutral” advantage. Similarly, distance traveled to the match is team specific; the home and away teams will not travel the same distance in general.

To add information that applies to both teams, you add a column to the match file with the column name being something descriptive, like ”surface”, ”format”, ”temperature”. To add information that is team specific, add ”home.” and ”away.” to the name of the information. For example, ”home.adv” and ”away.adv” or ”home.dist” and ”away.dist”. These must come in a ”home.” and ”away.” pair.

Once the match data has a column for the explanatory variables, you will be able to use that when you specify your model in the function `rank.teams`. To add the extra information to the match files, all the scrapers allow you to pass in extra variables. These are used to make the extra columns. The code below shows examples.

Scraping League Data

Demosphere

Demosphere provides a platform that is easy to scrape and is consistent across different leagues and tournaments. In addition, Demophere has a team number that can be inferred from the links and is often used across leagues, cups and years within a particular soccer association. That can be very helpful for figuring out what team is playing. The team name often varies cryptically between leagues, tournaments and cups so being able to determine the team number is helpful. Note the number only applies to a specific Demophere platform, e.g. teams in the Oregon Premier League have one number and would have a different one if they played in the Washington Cup, hosted by the Puget Sound Premier League, even though both organizations use Demosphere.

The function `scrape.demosphere` will scrape a Demosphere page with the schedule and write the data to a comma-delimited file (.csv) with columns date, home.team, home.score, away.team and away.score. The required arguments are the url and file. `url` is what to scrape and `file` is the name of the .csv file. If `get.surface=TRUE`, the surface information will also be scraped and added as a column named `surface`. You can pass in `date.format` to set the format of the dates in the date column. The default is 2012-12-31 style.

In the examples below, I will add columns `venue`, `surface`, `home.adv` and `away.adv` to the match files. `venue` specifies where the match was played. If you are scraping multiple leagues, you will want a column like this that specifies the league. You can call the column whatever you want. I use `venue` because I also will have tournament data and I put the tournament name in the `venue` column. It is wise to use a unique name for the league or venue. For example ”RCL D1” is not unique because there is a ”RCL D1” league at multiple ages and different seasons. ”Fall RCL D1 U11” or ”Fall'12 RCL D1 U11” would be unique. If you don't do this, you will run into non-unique league names when you get to new seasons or try to compare multiple ages. Here I use ”RCL D1” but in my real database, I use unique identifiers. `home.adv` and `away.adv` specify the advantage. For league games, I use ”home” for the home team and ”away” for the away team. For tournaments, I use ”neutral” for both teams.

When adapting this code, it is important that the url points to an analogous webpage. Go to the url and look at the format and then find the analogous page for your league. Demosphere has two styles of webpages. If you get errors after go get a message that surface information is being scraped, try get.surface=FALSE (or leave off since FALSE is the default).

```
# Update PSPL WSPL; tag will be the filename
# Try to get the surface information from the website
url = "http://www.pugetsoundpremierleague.com/schedules/Fall2012/48042146.html"
tag="PSPL WSPL"
file=tag
scrape.demosphere(url, file=file, venue=tag, get.surface=TRUE, home.adv="home", away.adv
    ="away")

# Update PSPL Classic; there's a different page for each month
# So I use append=TRUE for the subsequent months to indicate that data should be
   appended
tag="PSPL Classic"
file=tag
url = "http://www.pugetsoundpremierleague.com/schedules/Fall2012/48051962.20129.html"
scrape.demosphere(url, file=file, venue=tag, home.adv="home", away.adv="away")
url = "http://www.pugetsoundpremierleague.com/schedules/Fall2012/48051962.201210.html"
scrape.demosphere(url, file=file, venue=tag, append=TRUE, home.adv="home", away.adv="
   away")
url = "http://www.pugetsoundpremierleague.com/schedules/Fall2012/48051962.201211.html"
scrape.demosphere(url, file=file, venue=tag, append=TRUE, home.adv="home", away.adv="
   away")
url = "http://www.pugetsoundpremierleague.com/schedules/Fall2012/48051962.201212.html"
scrape.demosphere(url, file=file, venue=tag, append=TRUE, home.adv="home", away.adv="
   away")
```

The Oregon Premier League is scraped in the same way. Here I show how to scrape the NPL which has a page for each month and the division 4 league which has the whole schedule on one page.

```
# Update OPL NPL
# Again schedules for each month appear on different pages.
tag="OPL NPL"
file=tag
url = "http://www.oregonpremierleague.com/schedules/Fall2012/47896544.20129.html"
scrape.demosphere(url, file=file, venue=tag, home.adv="home", away.adv="away")
url = "http://www.oregonpremierleague.com/schedules/Fall2012/47896544.201210.html"
scrape.demosphere(url, file=file, venue=tag, append=TRUE, home.adv="home", away.adv="
   away")
url = "http://www.oregonpremierleague.com/schedules/Fall2012/47896544.201211.html"
scrape.demosphere(url, file=file, venue=tag, append=TRUE, home.adv="home", away.adv="
   away")

# Update OPL fourth division
# all schedules on one page
tag="OPL 4th Div"
file=tag
```

```
url = "http://www.oregonpremierleague.com/schedules/Fall2012/51445522.html"
scrape.demosphere(url, file=file, venue=tag, home.adv="home", away.adv="away")
```

The NorCal State Cup has a different style of webpage. Pass in table.style=2 and the year to scrape this page. You have to pass in the year because year is not included in the dates on the page. I set get.surface=FALSE because the scraper cannot interpret the Javascript code being used on the page.

```
tag="NorCal State Cup"
file=tag
url = "http://www.norcalpremier.com/schedules/59258181/59258272.html"
scrape.demosphere(url, file=file, venue=tag, get.surface=FALSE, home.adv="home", away.
   adv="away", table.style=2, year=2013)
```

Now we have a series of .csv files for these PSPL and OPL leagues and a NorCal State Cup. I put my match data in separate .csv files as I find it easier to find match data when I need to error check later, but you can just have one file name and append everything to that.

Korrio

Korrio is another content provider for soccer leagues and tournaments. It is a bit less standard than Demosphere and the scraper may need tweaking to work for other tournaments and leagues. The function is `scrape.korrio` and the arguments are the same as for `scrape.demosphere`.

Here I show how to scrape the RCL division 1 league. The division 2 to 4 leagues are the same; just a different url.

```
# Update RCL D1
tag="RCL D1"
file=tag
url="https://korrio.com/groups/division-1-2136679329/schedule"
scrape.korrio(url, file=file, venue=tag, get.surface=TRUE, home.adv="home", away.adv="
   away")
```

Affinity Sports

Affinity Sports is a commonly used content provider for USYSA soccer leagues and cups. This is a content provider that has different page designs for different leagues and tournaments. The argument `col.num` is the number of columns in the schedule table, and is used to try to determine which format is used. Look at the website and count all the columns. This scraper is unlikely to be stable across Affinity Sport sites, but give it a try. Just be ready to adapt the scraper code as needed. The function is `scrape.sportaffinity`. Affinity Sport sites do not typically have surface information for the fields, so I add `surface="Unk"` for unknown surface.

Here's how to scrape the Portland Metro League Red division.

```
#Portland Metro League
url="http://oysa-2012pml.sportsaffinity.com/Tour/public/info/schedule_results2.asp?
   sessionguid=&flightguid={EB53B0A5-1309-4884-9A96-BE0CD7C1307C}&tournamentguid=
   DD593D51-4F66-4C23-9784-F00DCD2FBC91"
tag="PML Blue"
file=tag
scrape.sportaffinity(url, file=file, venue=tag, col.num=10, home.adv="home", away.adv="
   away", surface="Unk")
```

The surface cannot be scraped. You'll have to look up the surface and edit the .csv file manually if you want surface information.

GotSport

GotSport (or GotSoccer) is another commonly used content provider, especially for tournaments but also some leagues. Unfortunately, this is a content provider that has different page designs for different leagues and tournaments. The argument `tb.num` which table on the page has the match data. I don't know how to get this by looking at the page. The `scrape.gotsport` function will return the list of tables if you use a bad `tb.num`. You can use this feature to get the list, and then look at the list to find the element with the match table. Here `tb.num=800` is way too big (it should be 8, 9 or 10), so this is guaranteed to throw and error and return the table list to you

```
url="http://events.gotsport.com/events/schedule.aspx?EventID=26296&GroupID=235574&Gender
   =Boys&Age=12"
my.tables=scrape.gotsport(url, file=file, venue=tag, tb.num=800, home.adv="home", away.
   adv="away", surface="Unk")
head(my.tables[[5]])
```

Look at `my.tables` and find which list element (the number) has the matches. That should be `tb.num`. You'll have to use trial and error to figure out what `tb.num` to use.

Note GotSport has a super nice feature of a universal team number. This number is used for any league or tournament the team is in. There are errors (like teams with multiple numbers) but in general the number is accurate and is very helpful for sorting out what team played in a GotSport tournament (since team name may be ambiguous or unhelpful).

The WA District VI leagues use GotSport. Here's how to scrape the Umbro league. The Reebok and Adidas leagues use the same `tb.num`.

```
# Update D6 Umbro
url="http://events.gotsport.com/events/schedule.aspx?EventID=26296&GroupID=235574&Gender
   =Boys&Age=12"
tag="D6 Umbro"
file=tag
a=scrape.gotsport(url, file=file, venue=tag, tb.num=5, home.adv="home", away.adv="away",
    surface="Unk")
```

Custom1: District III and NPSL

These leagues use webpages with no dates but rather with week number. Unfortunately, they also have malformed html tables. `scrape.custom1` will scrapes these and the function code will show you some tricks for dealing with malformed html.

Here is the code to scrape the North Puget Sound League (NPSL). The division 2 page has some games entered with the wrong week. The code shows how to fix that.

```
# You need to tell the code the dates for the week numbers.
weeks=c("2012-9-8","2012-9-15","2012-9-22","2012-9-29",
"2012-10-6","2012-10-13","2012-10-20","2012-10-27",
"2012-11-3","2012-11-10","2012-11-17","2012-12-1",
"2012-12-8","2012-12-15")

# Update NPSL D1
# if it throws an error, try changing first.td.tag to 4 or 2 or 5
url = "http://www.npsl-league.org/npsl/scorekeeping/score_report.php?report_id=B12D1"
tag="NPSL D1"
file=tag
scrape.custom1(url,weeks=weeks, file=file, venue=tag, first.td.tag=3, home.adv="home",
   away.adv="away", surface="Unk")

# Update NPSL D2
url = "http://www.npsl-league.org/npsl/scorekeeping/score_report.php?report_id=B12D2"
tag="NPSL D2"
file=tag
scrape.custom1(url,weeks=weeks, file=file, venue=tag, first.td.tag=3, home.adv="home",
   away.adv="away", surface="Unk")
#fix games that listed with wrong week
filename=paste(file,".csv",sep="")
scores=read.csv(file=filename, colClasses=c("character"),strip.white=TRUE)
scores$date[scores$date=="2012-12-8" & scores$home.team=="NSC Impact 00"]="2012-11-4"
scores$date[scores$date=="2012-12-15" & scores$away.team=="NSC Impact 00"]="2012-11-24"
write.table(scores, file=filename,row.names=FALSE,col.names=TRUE,append=FALSE,sep=",",
   qmethod="double")
```

JSON

The British Columbia Soccer Premier League (BCSPL) is an example of a league that uses Javascript to display match results and getting that dynamically produced html is hard (at least I was not able to figure out how to get it). But we can easily scrape the JSON from the BCSPL website. Unfortunately finding the url to the JSON is not so easy. See the url below and try to find the analogous one. [Update: the webpage has since changed, but I leave it here for reference. If you need to scrape JSON, try looking at my scrape.json1 code and adapt that.]

```
# Update BCSPL; finding the correct url is not easy; the # after /leagues/ is the
   important part
url="http://www.bcsoccerpremierleague.net/leagues/3483/events.json?calendar=true&amp;
   season_id=40085"
tag="BCSPL"
file=tag
scrape.json1(url, file=file, venue=tag, home.adv="home", away.adv="away", surface="Unk")
```

Scraping Tournament Data

Here I show a series of Pacific NW tournaments and the code I used to scrape them. In this example, I will append all the tournament data to one file called ”tournaments” (the function will add .csv if you leave it off). Notice I set `append=TRUE` to append to the existing file.

The Oswego Nike Cup is an example of a different GotSport format with an unusual date format. The date format is specified with `date.style`. It also has a different table style, which I call 2.

```
file="tournaments"
#this one has the table we want in tb.num=8
url="http://events.gotsport.com/events/schedule.aspx?EventID=24694&GroupID=234874&Gender
   =Boys&Age=12"
scrape.gotsport(url, file=file, venue="Oswego Nike Cup Gold", tb.num=8, home.adv="home",
    away.adv="away", surface="Unk")
#I use append=TRUE to add to the tournaments.csv file
url="http://events.gotsport.com/events/schedule.aspx?EventID=24694&GroupID=234875&Gender
   =Boys&Age=12"
tb=scrape.gotsport(url, file=file, venue="Oswego Nike Cup Silver", tb.num=8, append=TRUE
   , home.adv="home", away.adv="away", surface="Unk")
```

The Crossfire Nike Cup also uses GotSport but a different format and the table with the matches is in a different location. I set `surface=Grass"` since all these matches were held on Grass.

```
#xfire Nike
```

```
#this one has games in tb 12
#I keep append=TRUE to add to the tournaments.csv file
tag="Crossfire Nike Challenge Gold"
file="tournaments"
url="http://events.gotsport.com/events/schedule.aspx?EventID=23071&GroupID=232364&Gender
   =Boys&Age=13"
tb=scrape.gotsport(url,file=file, venue=tag, append=TRUE, tb.num=11, date.format="%m/%d
   /%Y", home.adv="neutral", away.adv="neutral", surface="Grass")
```

I assign the output to `tb` because webpage html is fluid and you might need to change `tb.num` to something different to make this work. The output if there is an error should give you a hint.

The Northwest Cup also uses GotSport but the table we need in number 10 not 12.

```
#Northwest Cup
url="http://events.gotsport.com/events/schedule.aspx?EventID=25578&GroupID=241973&Gender
   =Boys&Age=12"
tb=scrape.gotsport(url, file=file,venue="Northwest Cup Gold", tb.num=10, append=TRUE,
   home.adv="home", away.adv="away", surface="Unk")
```

The NW Champions League, Chinook Cup, River Jam and Rainier Challenge all use Demosphere.

```
# NWCL
url="http://www.northwestchampions.com/schedules/2012/48003952.html"
tag="NWCL"
file="tournaments"
scrape.demosphere(url, file=file, venue=tag, append=TRUE, home.adv="neutral", away.adv="
   neutral", surface="Unk")

#Chinook Cup Gold
file="tournaments"
url="http://southsidesoccer.org/schedules/2012/46586837.html"
scrape.demosphere(url, file=file, venue="Chinook Cup Gold", append=TRUE, home.adv="home
   ", away.adv="away", surface="Grass")
url="http://southsidesoccer.org/schedules/2012/46586838.html"
scrape.demosphere(url, file=file, venue="Chinook Cup Silver", append=TRUE, home.adv="
   home", away.adv="away", surface="Grass")
```

```
#River Jam
url="http://www.washingtonpremierfc.com/schedules/2012/47393723.html"
scrape.demosphere(url, file=file, venue="River Jam", append=TRUE, home.adv="home", away.
   adv="away", surface="Grass")

#Rainier Challenge
url="http://www.washingtonpremierfc.com/schedules/2012/45274431.html"
scrape.demosphere(url, file=file, venue="Rainer Challenge Samba", append=TRUE, home.adv
   ="home", away.adv="away", surface="Grass")
url="http://www.washingtonpremierfc.com/schedules/2012/45274432.html"
scrape.demosphere(url, file=file, venue="Rainer Challenge Copa", append=TRUE, home.adv="
   home", away.adv="away", surface="Grass")
```

At the end of this, you have a collection of match files that can form the basis of your match database for `rank_teams()`.