Using Optmatch on data in SAS, Stata, etc

Ben B. Hansen, Mark Fredrickson, Josh Buckner, Josh Errickson, and Peter Solenberger, with embedded Fortran code due to Dimitri P. Bertsekas and Paul Tseng

2016-05-02

General comments

If a users preferred data analysis software is other than R, Optmatch can still easily be used to perform the matching while all other data analysis can be performed in the preferred software.

In general, the procedure will be

1. In your preferred software, export a data set containing a treatment indicator and all variables to match, exactMatch or caliper on.
   - (Optionally, if you wish to match using a propensity score, fit such a model and include the predicted propensity scores in the data set.)
2. Import the data into R and perform the matching.
3. Export the data from R, including the matches.
4. Import the data back into your preferred software.

The two best ways to import the data back-and-forth are either using comma separated value files (.csv files) or using the R package `foreign`.

For .csv files, sample R code may be

```
> externaldata <- read.csv("externaldata.csv", header=TRUE)
> externaldata$match <- fullmatch(..., data=externaldata)
> write.csv(externaldata, file="externaldata.matched.csv")
```

Examples of the use of `foreign` are included within the software-specific sections below.

Using Optmatch with SAS

For this example, lets say we have some simple demographics. We will treat gender as the treatment indicator, and wish to match on a combination of a propensity score for gender (using both age and height) and age.

```
data people;
  infile datalines dsd dlm=' ' missover;
  input gender age height;

datalines;
0 25 62
0 41 68
0 38 63
0 22 62
1 33 70
1 35 71
1 47 68
1 23 64
;
run;
```

Now we can fit a logistic model to predict gender using age and height.

```
proc logistic data = people;
  model gender (event='1') = age height;
  output out = preddata p=ppty;
run;
```

Finally, since we want to match only on the new `ppty` propensity score and age, we can drop height.

```
proc data newpeople;
  set preddata;
  keep gender age ppty;
run;
```

With this setup, we can now either pass it to R via a .csv file, or directly using the foreign package in R.

Passing SAS data with .csv files

Save the file from SAS.

```
proc export data=newpeople;
  outfile="C:\Users\myuser\Desktop\sasout.csv";
run;
```

Inside R, we can load this data.

```
> sasdata <- read.csv("C:/Users/myuser/Desktop/sasout.csv", header=TRUE)
```

(Depending on your version of Windows, you may need to use

```
> sasdata <- read.csv("C:\\Users\\myuser\\Desktop\\sasout.csv", header=TRUE)
```

instead.)

If you have string variables (e.g. race as "White", "Hispanic", etc), you may need to include the argument `stringsAsFactors=FALSE`.

Now, perform matching as desired, saving the final match to `sasdata`. For example,

```
> library(optmatch)
> f <- fullmatch(gender ~ age + ppty, data=sasdata)
> sasdata$match <- f
```

Save this data back to .csv as follows.

```
> write.csv(sasdata, "C:/Users/myuser/Desktop/rout.sas.csv", row.names=FALSE)
```

The use of `row.names=FALSE` stops R from including the row names (likely 1, 2, 3, etc) as the first column in the data. If you re-arranged the data at any point, you may need to set that to `TRUE`, but keep in mind to handle it properly in SAS, as the default will be to treat it as a variable.

Now, returning to SAS, we can read the new rout.sas.csv file in. The only catch is that we want to ensure that the match is read as a string by using `$`, since it may have values like `1.1` and `1.10`, representing two different matches, but which are identical if treated as numeric.

```
data matchedpeople;
  infile "C:/Users/myuser/Desktop/rout.sas.csv" dsd firstobs=2;
  input gender age ppty match $;
run;
```

The argument `firstobs=2` skips the variable names; alternatively you could pass `col.names=FALSE` to R's `write.csv`, but then the rout.sas.csv file lacks any variable information, which may be useful to have.

Passing SAS data with R's foreign

As an alternative to using R's `write.csv`, you can use the R package foreign to generate both the data and SAS code needed.

```
> library(foreign)
> write.foreign(sasdata, 'C:/Users/myuser/Desktop/rout.sas.txt',
+               'C:/Users/myuser/Desktop/rout.code.sas', package = 'SAS')
```

Opening the rout.code.sas file will give you the SAS code to read in the data.

Using Optmatch with Stata

For this example, we will start with the built-in auto data set in Stata.

```
sysuse auto.dta
```

We will treat `foreign`, whether a car is domestic or foreign, as the treatment indicator. (Not to be confused with the R package foreign!) We will estimate propensity scores using all other variables (excluding `make` which is unique per row), and wish to match on the estimated propensity score as well as `price` and `mpg`.

First, lets fit the logistic regression model.

```
logit foreign price mpg rep78 headroom trunk weight length turn displacement gear_ratio
predict ppty
```

Now, we can export a .csv file. Note that in addition to the treatment indicator and variables to match on, we need to include a unique identifier. In this case, we can use `make`. If no such identifier exists, you can use

```
gen case_id = _n
```

to generate ID's 1, 2, etc. These will be needed to merge the match information back in.

```
outsheet make foreign price mpg ppty using C:\Users\myuser\Desktop\stataout.csv, comma
```

Turning to R, this can be read in similar to the SAS example, using

```
> statadata <- read.csv("C:/Users/myuser/Desktop/stataout.csv", header=TRUE)
```

(Again, depending on your version of Windows, you may need to use

```
> statadata <- read.csv("C:\\Users\\myuser\\Desktop\\stataout.csv", header=TRUE)
```

instead.)

If you have string variables (e.g. race as "White", "Hispanic", etc), you may need to include the argument `stringsAsFactors=FALSE`.

Now, perform matching as desired, saving the final match to `statadata`. For example,

```
> library(optmatch)
> f <- fullmatch(foreign ~ price + mpg + ppty, data=statadata, max.controls=3)
> statadata$match <- f
```

We do not recommend using .csv files to transfer the data back to Stata, though the `write.csv` file would be similar to that for SAS. Instead, we recommend using foreign.

```
> write.dta(statadata, "C:/Users/myuser/Desktop/rout.stata.dta")
```

Back in Stata, you can merge this into the existing `auto.dta` by the following commands.

```
sort make
merge 1:1 make using "C:/Users/myuser/Desktop/rout.stata.dta", force
```

The `force` option is necessary to overcome type differences. Additional tweaks may be necessary here if you have special variable types.