MatchingFrontier: Computing the Balance-Sample Size Frontier in Matching Methods

Gary King
Harvard University

Christopher Lucas
Harvard University

Richard Nielsen
MIT

Abstract

MatchingFrontier implements the methods described in King, Lucas, and Nielsen (n.d.) for optimizing both balance and sample size. **MatchingFrontier** supports the computation of frontiers for both continuous and discrete metrics, and also provides functions for visualizing the frontier. Additional functionality to be added in the coming months.

Keywords: R, matching, frontier, Mahalanobis, L1.

1. About MatchingFrontier

2. Installation

You can install the latest stable version of **MatchingFrontier** with `install.packages('MatchingFrontier', repos = 'http://cran.us.r-project.org')`. If you'd prefer to use the development version, you may do so by installing the package from the public-facing Github repository. We recommend using the **devtools** package to do so, as follows.

```
library(devtools)
install_github('MatchingFrontier')
```

3. Using MatchingFrontier

At present, the **MatchingFrontier** namespace includes just a few functions (feedback on additional functionality is greatly appreciated). Those functions support the computation of the frontier, the estimation of effects along it, plotting functions, and support for exporting optimal data sets on the balance - sample size frontier. We will illustrate the use of **MatchingFrontier** with the commonly used Lalonde data, which is included in the package.

The user must first create the frontier. To do so, use the `makeFrontier()` function, as follows.

```
data('lalonde')
match.on <- colnames(lalonde)[!(colnames(lalonde) %in% c('re78', 'treat'))]
my.frontier <- makeFrontier(dataset = lalonde,
                            treatment = 'treat',
                            outcome = 're78',
                            match.on = match.on)
```

`match.on` is a character vector holding the variable names that the user wishes to match on. In the Lalonde data, `re78` is the outcome and `treat` is the treatment, so we exclude those variable names from the character vector.

By default, `makeFrontier()` calculates the frontier for the Average Mahalanobis Distance metric, primarily because this is the fastest at present. The quantity of interest is the *feasible sample average treatment effect on the treated* or FSATT. Weights are later computed in the estimation stage. The full function is as follows. For complete information, see the help file in the package with `?makeFrontier`.

```
  makeFrontier(dataset,
               treatment,
               outcome,
               match.on, QOI = 'FSATT',
               metric = 'Mahal',
               ratio = 'variable',
               breaks = NULL)
```

Continuing with the Lalonde example, next we'll estimate the effects along the frontier with the `estimateEffects()` function, which takes the output from `makeFrontier` to estimate the effect of the treatment along all values of the frontier. This can be quite slow for the obvious reason that if there exist thousands or tens of thousands of points on the frontier, the code can be no faster than the time it takes to estimate a single effect times the total number of points (which quickly becomes quite long). Very soon, we will support estimating a random sample of points along the frontier to reduce computation time.

With the Lalonde example, the code is as follows.

```
my.estimates <- estimateEffects(my.frontier,
                                're78 ~ treat')
```

The first argument is the output from `makeFrontier` and the second is the formula passed to the `lm()` function. `estimateEffects` stores the estimates and the 95% confidence interval for each point it estimates.

Next, we can plot the frontier and the estimates with the plotting functions, as follows.

```
# Plot frontier
plotFrontier(my.frontier)

# Plot estimates
plotEstimates(my.frontier, my.estimates)
```

Both plotting functions use the ellipses feature to pass arguments to the base `plot` function, so any argument that one can use with `plot()` can also be used with `plotFrontier()` and `plotEstimates()`, such as 'main' for the title, 'xlab' and 'ylab' for the axis labels, etc.

Lastly, users may wish to export a data set on the frontier for additional analysis. To do so, use the `generateDataset()` function, as follows.

```
n = 1000 # Identify the point from which to select the data
generateDataset(my.frontier, N = n)
```

If the estimand is variable ratio, as it is by default, the exported data set will include the appropriate weights necessary for estimating the FSATT.

Affiliation:

Gary King
Department of Government
Harvard University
1737 Cambridge St, Cambridge, MA, USA
E-mail: king@harvard.edu
URL: http://gking.harvard.edu/

Christopher Lucas
Department of Government
Harvard University
1737 Cambridge St, Cambridge, MA, USA
E-mail: clucas@fas.harvard.edu
URL: christopherlucas.org

Richard Nielsen
Department of Political Science
Massachusetts Institute of Technology
77 Massachusetts Avenue, Cambridge, MA, USA
E-mail: rnielsen@mit.edu
URL: http://www.mit.edu/ rnielsen/index.htm