MatchingFrontier: Automated Matching for Causal Inference*

Gary King
Harvard University

Christopher Lucas
Harvard University

Richard Nielsen
MIT

Abstract

MatchingFrontier is an R package that implements the methods described in King, Lucas, and Nielsen (n.d.) for simultaneously optimizing both balance and sample size in matching methods for causal inference. **MatchingFrontier** supports the computation of frontiers for both continuous and discrete metrics and also provides functions for visualizing the frontier and exporting matched data sets for further analysis.

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

1. Introduction

Matching methods have become extremely popular amongst researchers working with observational data, especially when used as a nonparametric preprocessing step to reduce model dependence (Ho, Imai, King, and Stuart 2007, 2009). But despite this popularily, existing matching approaches leave researchers with two fundamental tensions. First, they are designed to maximize one metric (such as propensity score or Mahalanobis distance) but are judged against another for which they were not designed (such as $L1$ or differences in means). Second, they lack a principled solution to revealing the implicit bias-variance trade off: matching methods need to optimize with respect to both imbalance (between the treated and control groups) and the number of observations pruned, but existing approaches optimize with respect to only one; users then either ignore the second or tweak it without a formal stopping rule.

MatchingFrontier resolves both tensions by consolidating previous techniques into a single, optimal, and flexible approach. The software calculates the matching solution with maximum balance for each possible sample size $(N, N-1, N-2, ...)$ and returns each solution, the whole of which constitute the *frontier*, from which the user can easily choose one, several, or all subsamples with which to conduct the final analysis, given their own choice of imbalance metric and quantity of interest. **MatchingFrontier** solves the joint optimization problem in one run, automatically, without manual tweaking, and without iteration. Although for each subset size k, there exist a huge number of unique subsets $\binom{N}{k}$, **MatchingFrontier** includes specially designed and extremely fast algorithms that give the optimal answer, usually in a few minutes or less.

*The current release of **MatchingFrontier** is in active development and will continue to grow over the coming months. Comments and suggestions are greatly appreciated.

2. General Framework

Matching methods are designed to reduce imbalance in data by selectively pruning observations, which in turn reduces model dependence (King and Zeng 2006; Imai, King, and Stuart 2008; Iacus, King, and Porro 2011b; Ho *et al.* 2007). However, pruning reduces sample size and therefore may increase variance in the eventual estimates. Users of matching are then confronted with the perennial bias-variance trade-off. Perhaps surprisingly, existing approaches to matching do not conduct the implied joint optimization of bias and variance. Rather, they improve one dimension of the optimization and leave the second to the user. Such an approach is time consuming and rarely yields the optimal solution.

King *et al.* (n.d.) proposes a solution to this joint optimization, which is implemented in **MatchingFrontier**. Discrete and continuous metrics are defined and algorithms are provided for both continuous and discrete metrics, thus rendering the method agnostic to the metric. We point users of **MatchingFrontier** to King *et al.* (n.d.) for algorithmic details and theoretical proofs. In this section, we provide definitions of the metrics so that users can choose appropriately when using `makeFrontier()`.

For discrete metrics, we follow (Iacus, King, and Porro 2011a) and use the difference between the multivariate histograms of the treated and control groups. Formally, let $f_{\ell_1 \cdots \ell_k}$ be the relative empirical frequency of treated units in a bin with coordinates on each of the X variables as $\ell_1 \cdots \ell_k$ so that $f_{\ell_1 \cdots \ell_k} = n_{T_{\ell_1 \cdots \ell_k}} / n_T$ where $n_{T_{\ell_1 \cdots \ell_k}}$ is the number of treated units in stratum $\ell_1 \cdots \ell_k$ and n_T is the number of treated units in all strata. We define $g_{\ell_1 \cdots \ell_k}$ similarly among control units. Then:

$$L_1(H) = \frac{1}{2} \sum_{(\ell_1 \cdots \ell_k) \in H} |f_{\ell_1 \cdots \ell_k} - g_{\ell_1 \cdots \ell_k}| \tag{1}$$

For continuous metrics, we define the Average Mahalanobis Imbalance (AMI). Though easily generalized to all continuous measures of distance, we choose Mahalanobis distance. AMI is the distance between each unit i and the closest unit in the opposite group, averaged over all units: $D = \text{mean}_i[D(X_i, X_{j(i)})]$, where the closest unit in the opposite group is $X_{j(i)} = \arg\min_{X_j | j \in \{1-T_i\}} [D(X_i, X_j)]$ and $\{1 - T_i\}$ is the set of units in the (treatment or control) group that does not contain i. **MatchingFrontier** defaults to AMI but can just as easily be used with L_1.

Of note is that these metrics presume a dichotomous treatment. Given recent advances in matching with continuous treatments (Iacus and King n.d.; Ratkovic n.d.), we encourage researchers to consider generalizing our algorithms (and therefore, metrics) to continuous treatment regimes.

3. Getting Started

MatchingFrontier is written in the R language (Team *et al.* 2012) and is currently hosted on Github and CRAN. CRAN hosts the latest stable release. You can install the current development release of **MatchingFrontier** with the **devtools** package (Wickham and Chang 2013), as follows.

```
> library(devtools)
> install_github('ChristopherLucas/MatchingFrontier')
```

Alternatively, you can install the development version of **MatchingFrontier** from a *nix command line as follows.

```
$ curl -OL https://github.com/ChristopherLucas/MatchingFrontier/archive/master.zip
$ unzip master.zip
$ cd MatchingFrontier-master
$ R CMD INSTALL package
```

4. A User's Guide

The typical **MatchingFrontier** workflow is displayed in Figure 1. Note that in nearly all cases, users first proceed through the two-step process of computing the frontier and then estimating quantities of interest across it. After these steps are completed, the results can be used to visually summarize the full frontier or to closely inspect a particular point on it. Next, we illustrate this workflow with the LaLonde data (LaLonde 1986; Dehejia and Wahba 1999), which is included in **MatchingFrontier**.

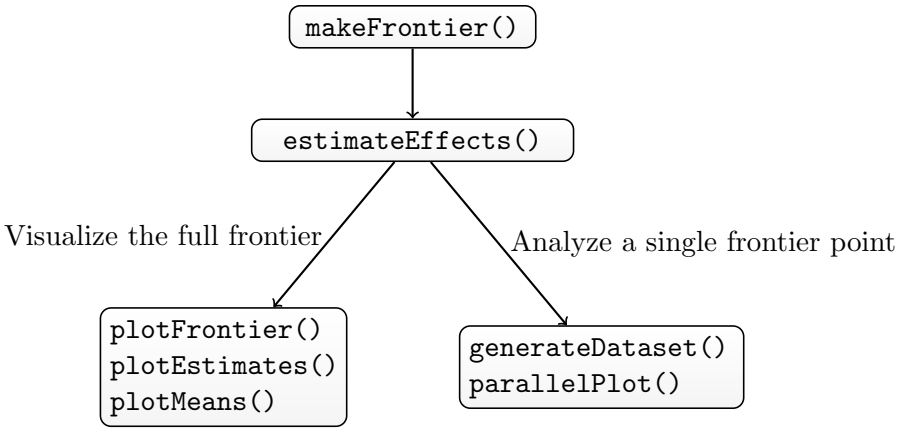

Figure 1: A typical **MatchingFrontier** workflow. `makeFrontier()` is used to construct the frontier, then `estimateEffects()` is used to estimate quantities of interest for each point on the frontier, after which the user may proceed to visualize the full frontier or to inspect individual points on it.

4.1. LaLonde Example

For the running example in this paper, we use a randomly selected subset of the "LaLonde" data (LaLonde 1986; Dehejia and Wahba 1999)[1]. The LaLonde data is commonly used to assess matching methods and refers to the combination of data from an experimental intervention containing 185 treated units (the National Supported Work Demonstration) with observational data. By combining the experimental data with observational data, methods can be compared to the underlying experimental benchmark. We follow LaLonde (1986) and

[1]For a complete description of the data, type `?lalonde` after loading **MatchingFrontier**.

combine the results of the experimental intervention with the Current Population Survey. In King *et al.* (n.d.), we analyze the Lalonde data plus the full data set from the Current Population Study. In this paper, we keep the 185 treated units and randomly selected 1,000 controls from the full data. This allows users to quickly replicate and adapt the code presented in this paper. See King *et al.* (n.d.) for a serious substantive analysis.

The LaLonde data contains a treatment indicator "treat" (an indicator for assigment to a jobs training program), an outcome measure "re78" (income in 1978), and a number of controls (potential confounders) that we will match on during the illustration. The controls are as follows.

age: subject age at time of intervention

education: years of education

black: a race indicator for identification as black

hispanic: an ethnicity indicator for identification as hispanic

married: an indicator for whether or not the subject is married

nodegree: an indicator for whether or not the subject has a college degree

re74: income in 1974

re75: income in 1975

4.2. Computing the Frontier

The user must first create the frontier. To do so, use the `makeFrontier()` function, which will calculate the optimal subsample at every point on the frontier. By default, `makeFrontier()` calculates the frontier with the Average Mahalanobis Imbalance. However, as we demonstrate, **MatchingFrontier** works just as easily with L_1 difference.

First, calculate the Mahalanobis frontier for the LaLonde data.

```
> # Load the package and the data
> library(MatchingFrontier)
> data('lalonde')
> # Create a vector of column names to indicate which variables we
> # want to match on. We will match on everything except the treatment
> # and the outcome.
> match.on <- colnames(lalonde)[!(colnames(lalonde) %in% c('re78', 'treat'))]
> match.on # Print variables in match.on

[1] "age"       "education" "black"     "hispanic"  "married"   "nodegree"
[7] "re74"      "re75"
```

```
> # Make the mahalanobis frontier
> mahal.frontier <- makeFrontier(dataset = lalonde,
+                                treatment = 'treat',
+                                outcome = 're78',
+                                match.on = match.on)

Calculating Mahalanobis distances...
Calculating theoretical frontier...
Calculating information for plotting the frontier...

> mahal.frontier

An imbalance frontier with 997 points.
```

As shown above, `match.on` is a vector holding the variable names that the user wishes to match on. Because `re78` is the outcome and `treat` is the treatment, we exclude those variable names from the vector.

By default, `makeFrontier()` calculates the frontier for the Average Mahalanobis Imbalance, as defined in Section 2. The default quantity of interest is the *feasible sample average treatment effect on the treated* or FSATT (King *et al.* n.d.), for which weights are calculated and returned to the user.

To instead calculate the L_1 frontier, simply provide optional "metric", "QOI", and "ratio" arguments, as follows.[2]

```
> # Make the L1 frontier
> L1.frontier <- makeFrontier(dataset = lalonde,
+                             treatment = 'treat',
+                             outcome = 're78',
+                             match.on = match.on,
+                             QOI = 'SATT',
+                             metric = 'L1',
+                             ratio = 'fixed')

Calculating L1 binnings...
Calculating L1 frontier... This may take a few minutes...

> L1.frontier

An imbalance frontier with 976 points.
```

Next, we will use the results computed above to estimate causal effects along the frontier.

4.3. Estimating Effects

Continuing with the Lalonde example, we will estimate the effects along the frontier with the `estimateEffects()` function, which takes the output from `makeFrontier()` to estimate the

[2]For technical explanations of these arguments, we point users to King *et al.* (n.d.).

effect of the treatment along all values of the frontier. With the Lalonde example, the code is as follows.

```
> # Estimate effects for the mahalanobis frontier
> mahal.estimates <- estimateEffects(mahal.frontier, 're78 ~ treat')
> # Estimate effects for the L1 frontier
> L1.estimates <- estimateEffects(L1.frontier, 're78 ~ treat',
+                                 model.dependence.points = 100)
```

`estimateEffects()` takes two arguments. 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.

Alternatively, we could also condition on the variables that we are matching on (stored in `match.on`) by specifying a different formula, as follows.

```
> # Estimate effects for the mahalanobis frontier
> mahal.estimates.controls <-
+     estimateEffects(mahal.frontier, paste('re78 ~ treat +',
+                                           paste(match.on, collapse = ' + ')))
> # Estimate effects for the L1 frontier
> L1.estimates.controls <-
+     estimateEffects(L1.frontier, paste('re78 ~ treat +',
+                                        paste(match.on, collapse = ' + ')))
```

We've now estimated effects along the full frontier for AMI with and without controls and for L_1 with and without controls. Next, we will visually inspect the full frontier.

4.4. Plotting the Frontier

We can plot the frontier and the estimates with the plotting functions, as follows. Note that for the sake of brevity, we will only do so with the $L1$ frontier (no controls). However, to plot the other three frontiers calculated in the previous section, simply pass the corresponding objects to the plotting functions, as the syntax is the same.

First, we will plot the frontier, where the y-axis is $L1$ and the x-axis is the number of observations pruned. This is displayed in Figure 2 next to the code that generated it.

```
> # Plot frontier
> plotFrontier(L1.frontier)
```

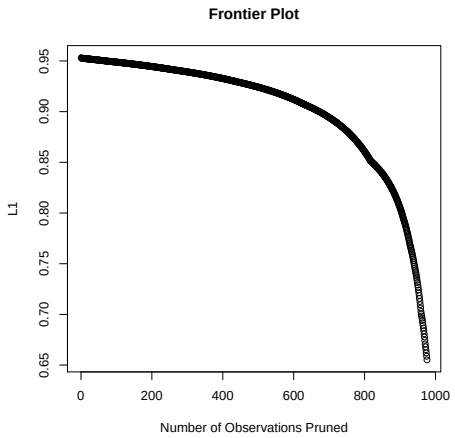

Figure 2: The L1 frontier without optional arguments

However, Figure 2 is not especially attractive. The font is too small and the dots constituting the frontier run into each other and create an ugly, fat line. All of the plotting functions in **MatchingFrontier** use R's ellipsis feature to permit access to the base plotting functionality. Figure 3 shows an example, along with the corresponding code.

```
> # Plot frontier
> plotFrontier(L1.frontier,
+              cex.lab = 1.4,
+              cex.axis = 1.4,
+              type = 'l',
+              panel.first =
+                 grid(NULL,
+                      NULL,
+                      lwd = 2)
+              )
```

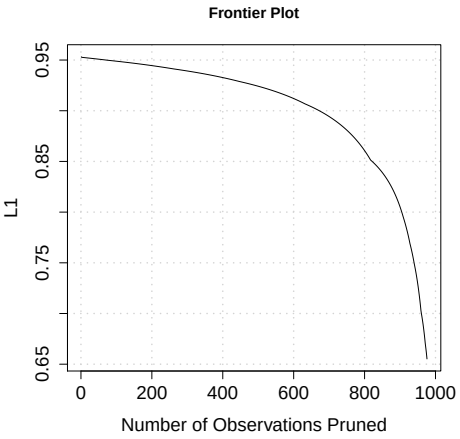

Figure 3: The $L1$ frontier with optional arguments

4.5. Plotting Estimates

Next, we can plot estimates along the frontier. As in the previous section, we will use the $L1$ frontier without controls. To do so, we'll use the results from `makeFrontier()` and `frontierEst()`. Figure 4 displays these results.

```
> # Plot estimates
> plotEstimates(L1.estimates,
+               ylim =
+                   c(-10000, 3000),
+               cex.lab = 1.4,
+               cex.axis = 1.4,
+               type = 'l',
+               panel.first =
+                  grid(NULL,
+                       NULL,
+                       lwd = 2,
+                       )
+               )
```

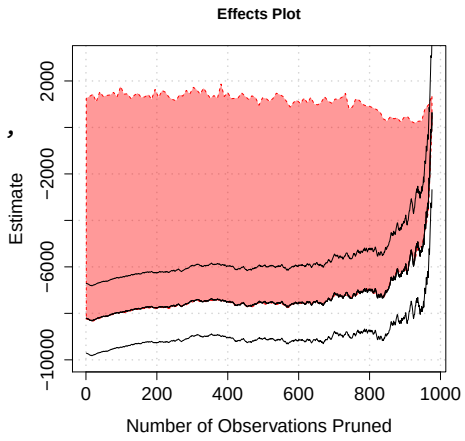

Figure 4: Estimates across the $L1$ frontier.

4.6. Inspect a Single Point on the Frontier

Lastly, users may wish to export a data set on the frontier for additional analysis. To do so, users are likely to rely on `parallelPlot()` and `generateDataset()`. Parellel plot allows the user to visually inspect multiple dimensions of a data set and requires only the output of `makeFrontier()`. For illustration, we will create a parallel plot that displays the treated and control values on 'age', 're74', 're75', and 'black' for the point on the frontier where 785 observations have been dropped. We will color treated units blue and control units gray.

```
> # Make parallel plot
> parallelPlot(L1.frontier,
+              N = 400,
+              variables = c('age',
+              're74',
+              're75',
+              'black'),
+              treated.col = 'blue',
+              control.col = 'gray'
+              )
```

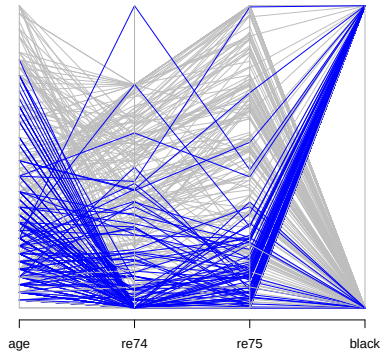

Figure 5: Parallel plot for pruning 785 observations.

Figure 5 makes obvious the fact that there are many more control than treated units and that the sample still contains a large number of controls that are not good matches for treated units, at least on these dimensions. Though this implies that perhaps we might move even further down the frontier, for illustration, let's now export this data set, using `generateDataset()` as follows.

```
> n <- 400 # Identify the point from which to select the data
> matched.data <- generateDataset(L1.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. We can now run a few simple regressions, controlling for the variables we matched on, using the matched data.[3]

5. Conclusion

We demonstrated how to use the new R software package **MatchingFrontier** for causal inference with observational data. With the LaLonde data, users were shown how to compute the balance-sample size frontier, calculate estimates along it, and visualize and inspect the results.

[3]Table generated by **Stargazer** (Hlavac 2014).

Table 1:

	Dependent variable:	
	re78	re78
	(1)	(2)
treat	−6,984.022***	1,011.661
	(886.364)	(1,120.759)
age		−14.088
		(44.293)
education		306.848
		(213.445)
black		−550.212
		(1,091.106)
hispanic		−1,508.171
		(1,447.458)
married		441.041
		(974.394)
nodegree		−489.522
		(1,165.794)
re74		0.115
		(0.093)
re75		0.566***
		(0.098)
Constant	13,333.170***	2,234.911
	(602.793)	(3,304.559)
Observations	400	400
R^2	0.135	0.402
Adjusted R^2	0.133	0.388
Residual Std. Error	8,838.675 (df = 398)	7,425.526 (df = 390)
F Statistic	62.085*** (df = 1; 398)	29.096*** (df = 9; 390)
Note:		*p<0.1; **p<0.05; ***p<0.01

References

Dehejia RH, Wahba S (1999). "Causal effects in nonexperimental studies: Reevaluating the evaluation of training programs." *Journal of the American statistical Association*, **94**(448), 1053–1062.

Hlavac M (2014). "stargazer: LaTeX code and ASCII text for well-formatted regression and summary statistics tables." *R package version 5.1*. URL http://CRAN.R-project.org/package=stargazer.

Ho DE, Imai K, King G, Stuart EA (2007). "Matching as nonparametric preprocessing for reducing model dependence in parametric causal inference." *Political analysis*, **15**(3), 199–236.

Ho DE, Imai K, King G, Stuart EA (2009). "MatchIt: nonparametric preprocessing for parametric causal inference." *Journal of Statistical Software*.

Iacus SM, King G (n.d.). "How coarsening simplifies matching-based causal inference theory."

Iacus SM, King G, Porro G (2011a). "Causal inference without balance checking: Coarsened exact matching." *Political analysis*, p. mpr013.

Iacus SM, King G, Porro G (2011b). "Multivariate matching methods that are monotonic imbalance bounding." *Journal of the American Statistical Association*, **106**(493), 345–361.

Imai K, King G, Stuart EA (2008). "Misunderstandings between experimentalists and observationalists about causal inference." *Journal of the royal statistical society: series A (statistics in society)*, **171**(2), 481–502.

King G, Lucas C, Nielsen RA (n.d.). "The Balance-Sample Size Frontier in Matching Methods for Causal Inference." *Working Paper*.

King G, Zeng L (2006). "The dangers of extreme counterfactuals." *Political Analysis*, **14**(2), 131–159.

LaLonde RJ (1986). "Evaluating the econometric evaluations of training programs with experimental data." *The American Economic Review*, pp. 604–620.

Ratkovic M (n.d.). "A Matching Method for General Treatment Regimes."

Team RC, *et al.* (2012). "R: A language and environment for statistical computing."

Wickham H, Chang W (2013). "devtools: Tools to make developing R code easier." *R package version*, **1**(1).

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