MatchIt: Nonparametric Preprocessing for Parametric Causal Inference[1]

Daniel E. Ho,[2] Kosuke Imai,[3] Gary King,[4] Elizabeth A. Stuart[5]

April 26, 2011

[1]We thank Olivia Lau for helpful suggestions about incorporating MatchIt into Zelig.

[2]Assistant Professor of Law & Robert E. Paradise Faculty Scholar, Stanford Law School (559 Nathan Abbott Way, Stanford CA 94305; http://dho.stanford.edu, dho@law.stanford.edu, (650) 723-9560).

[3]Assistant Professor, Department of Politics, Princeton University (Corwin Hall 041, Department of Politics, Princeton University, Princeton NJ 08544, USA; http://imai.princeton.edu, kimai@Princeton.Edu).

[4]David Florence Professor of Government, Harvard University (Institute for Quantitative Social Science, 1737 Cambridge Street, Harvard University, Cambridge MA 02138; http://GKing.Harvard.Edu, King@Harvard.Edu, (617) 495-2027).

[5]Assistant Professor, Departments of Mental Health and Biostatistics, Johns Hopkins Bloomberg School of Public Health (624 N Broadway, Room 804, Baltimore, MD 21205; http://www.biostat.jhsph.edu/∼estuart, estuart@jhsph.edu).

Contents

Chapter 1

Introduction

1.1 What MatchIt Does

MatchIt implements the suggestions of Ho, Imai, King, and Stuart (2007) for improving parametric statistical models and reducing model dependence by preprocessing data with semi-parametric and non-parametric matching methods. After appropriately preprocessing with MatchIt, researchers can use whatever parametric model and software they would have used without MatchIt, without other modification, and produce inferences that are more robust and less sensitive to modeling assumptions. (In addition, you may wish to use Zelig (`http://gking.harvard.edu/zelig/`; Imai et al. 2006 for subsequent parametric analyses, as it is designed to be convenient in analyzing MatchIt data sets.) MatchIt reduces the dependence of causal inferences on commonly made, but hard-to-justify, statistical modeling assumptions via the largest range of sophisticated matching methods of any software we know of. The program includes most existing approaches to matching and even enables users to access methods implemented in other programs through its single, unified, and easy-to-use interface. In addition, we have written MatchIt so that adding new matching methods to the software is as easy for anyone with the inclination as it is for us.

1.2 Software Requirements

MatchIt works in conjunction with the R programming language and statistical software, and will run on any platform where R is installed (Windows, Unix, or Mac OS X). R is available free for download at the Comprehensive R Archive Network (CRAN) at http://cran.r-project.org/. MatchIt has been tested on the most recent version of R. A good way to learn R, if you don't know it already, is to learn Zelig (available at http://gking.harvard.edu/zelig) which includes a self-contained introduction to R and can be used to analyze the matched data after running MatchIt.

1.3 Installing MatchIt

To install MatchIt for all platforms, type at the R command prompt,

```
> install.packages("MatchIt")
```

and MatchIt will install itself onto your system automatically. (During the installation process you may either decide to keep or discard the installation files, which will not affect the way MatchIt runs.)

1.4 Loading MatchIt

You need to install MatchIt only once, but you must load it prior to each use. You can do this at the R prompt:

```
> library(MatchIt)
```

Alternatively, you can specify R to load MatchIt automatically at launch by editing the `Rprofile` file located in the R program subdirectory, e.g. `C:/R/rw2011/etc/`, for Windows systems or the `.Rprofile` file located in the home directory for Unix/Linux and Mac OS X systems, and adding this line:

```
options(defaultPackages = c(getOption("defaultPackages"), "MatchIt"))
```

For this change to take effect, you need to restart R.

1.5 Updating MatchIt

We recommend that you periodically update MatchIt at the R prompt by typing:

```
> update.packages()
> library(MatchIt)
```

which will update all the libraries including MatchIt and load the new version of MatchIt.

Chapter 2

Statistical Overview

MatchIt is designed for causal inference with a dichotomous treatment variable and a set of pretreatment control variables. Any number or type of dependent variables can be used. (If you are interested in the causal effect of more than one variable in your data set, run MatchIt separately for each one; it is unlikely in any event that any one parametric model will produce valid causal inferences for more than one treatment variable at a time.) MatchIt can be used for other types of causal variables by dichotomizing them, perhaps in multiple ways (see also Imai and van Dyk 2004). MatchIt works for experimental data, but is usually used for observational studies where the treatment variable is not randomly assigned by the investigator, or the random assignment goes awry.

We adopt the same notation as in Ho, Imai, King, and Stuart (2007). Unless otherwise noted, let i index the n units in the data set, n_1 denote the number of treated units, n_0 denote the number of control units (such that $n = n_0 + n_1$), and x_i indicate a vector of pretreatment (or control) variables for unit i. Let $t_i = 1$ when unit i is assigned treatment, and $t_i = 0$ when unit i is assigned control. (The labels "treatment" and "control" and values 1 and 0 respectively are arbitrary and can be switched for convenience, except that some methods of matching are keyed to the definition of the treated group.) Denote $y_i(1)$ as the potential outcome of unit i under treatment — the value the outcome variable would take if t_i were equal to 1, whether or not t_i in fact is 0 or 1 – and $y_i(0)$ the potential outcome of unit i under control — the value the outcome variable would take if t_i were equal to 0, regardless of its value in fact. The variables $y_i(1)$ and $y_i(0)$ are jointly unobservable, and for each i, we observe one $y_i = t_i y_i(1) + (1 - t_i) y_i(0)$, and not the other.

Also denote a fixed vector of exogenous, pretreatment measured confounders as X_i. These variables are defined in the hope or under the assumption that conditioning on them appropriately will make inferences ignorable. Measures of balance should be computed with respect to all of X, even if some methods of matching only use some components.

2.1 Preprocessing via Matching

If t_i and X_i were independent, we would not need to control for X_i, and any parametric analysis would effectively reduce to a difference in means of Y for the treated and control groups. The goal of matching is to preprocess the data prior to the parametric analysis so that the actual relationship between t_i and X_i is eliminated or reduced without introducing bias and or increasing inefficiency too much.

When matching we select, duplicate, or selectively drop observations from our data, and we do so without inducing bias as long as we use a rule that is a function only of t_i and X_i and does not depend on the outcome variable Y_i. Many methods that offer this preprocessing are included here, including exact, subclassification, nearest neighbor, optimal, and genetic matching. For many of these methods the propensity score–defined as the probability of receiving the treatment given the covariates–is a key tool. In order to avoid changing the quantity of interest, most MatchIt routines work by retaining all treated units and selecting (or weighting) control units to include in the final data set; this enables one to estimate the average treatment effect on the treated (the purpose of which is described in Section 2.3).

MatchIt implements and evaluates the choice of the rules for matching. Matching sometimes increases efficiency by eliminating heterogeneity or deleting observations outside of an area where a model can reasonably be used to extrapolate, but one needs to be careful not to lose too many observations in matching or efficiency will drop more than the reduction in bias that is achieved.

The simplest way to obtain good matches (as defined above) is to use one-to-one exact matching, which pairs each treated unit with one control unit for which the values of X_i are identical. However, with many covariates and finite numbers of potential matches, sufficient exact matches often cannot be found. Indeed, many of the other methods implemented in MatchIt attempt to balance the overall covariate distributions as much as possible, when sufficient one-to-one exact matches are not available.

A key point in Ho, Imai, King, and Stuart (2007) is that matching methods by themselves are not methods of estimation: Every use of matching in the literature involves an analysis step following the matching procedure, but almost all analyses use a simple difference in means. This procedure is appropriate only if exact matching was conducted. In almost all other cases, some adjustment is required, and there is no reason to degrade your inferences by using an inferior method of analysis such as a difference in means even when improving your inferences via preprocessing. Thus, with MatchIt, you can improve your analyses in two ways. MatchIt analyses are "doubly robust" in that if *either* the matching analysis *or* the analysis model is correct (but not necessarily both) your inferences will be statistically consistent. In practice, the modeling choices you make at the analysis stage will be much less consequential if you match first.

2.2 Checking Balance

The goal of matching is to create a data set that looks closer to one that would result from a perfectly blocked (and possibly randomized) experiment. When we get close, we break the link between the treatment variable and the pretreatment controls, which makes the parametric form of the analysis model less relevant or irrelevant entirely. To break this link, we need the distribution of covariates to be the same within the matched treated and control groups.

A crucial part of any matching procedure is, therefore, to assess how close the (empirical) covariate distributions are in the two groups, which is known as "balance." Because the outcome variable is not used in the matching procedure, any number of matching methods can be tried and evaluated, and the one matching procedure that leads to the best balance can be chosen. MatchIt provides a number of ways to assess the balance of covariates after matching, including numerical summaries such as the "mean Diff." (difference in means) or the difference in means divided by the treated group standard deviation, and summaries based on quantile-quantile plots that compare the empirical distributions of each covariate. The widely used procedure of doing t-tests of the difference in means is highly misleading and should never be used to assess balance; see Imai et al. (2008).

These balance diagnostics should be performed on all variables in X, even if some are excluded from one of the matching procedures.

2.3 Conducting Analyses after Matching

The most common way that parametric analyses are used to compute quantities of interest (without matching) is by (statistically) holding constant some explanatory variables, changing others, and computing predicted or expected values and taking the difference or ratio, all by using the parametric functional form. In the case of causal inference, this would mean looking at the effect on the expected value of the outcome variable when changing T from 0 to 1, while holding constant the pretreatment control variables X at their means or medians. This, and indeed any other appropriate analysis procedure, would be a perfectly reasonable way to proceed with analysis after matching. If it is the chosen way to proceed, then either treated or control units may be deleted during the matching stage, since the same parametric structure is assumed to apply to all observations.

In other instances, researchers wish to reduce the assumptions inherent in their statistical model and so want to allow for the possibility that their treatment effect to vary over observations. In this situation, one popular quantity of interest used is the *average treatment effect on the treated* (ATT). For example, for the treated group, the potential outcomes under control, $Y_i(0)$, are missing, whereas the outcomes under treatment, $Y_i(1)$, are observed, and the goal of the analysis is to impute the missing outcomes, $Y_i(0)$ for observations with $T_i = 1$. We do this via simulation using a parametric statistical model such as regression, logit, or others (as described below). Once those potential outcomes are imputed from the model, the estimate of individual i's treatment effect is $Y_i(1) - \widehat{Y}_i(0)$ where $\widehat{Y}_i(0)$ is a predicted value of

the dependent variable for unit i under the counterfactual condition where $T_i = 0$. The in-sample average treatment effect for the treated individuals can then be obtained by averaging this difference over all observations i where in fact $T_i = 1$. Most MatchIt algorithms retain all treated units, and choose some subset of or repeated units from the control group, so that estimating the ATT is straightforward. If one chooses options that allow matching with replacement, or any solution that has different numbers of controls (or treateds) within each subclass or strata (such as full matching), then the parametric analysis following matching must accomodate these procedures, such as by using fixed effects or weights, as appropriate. (Similar procedures can also be used to estimate various other quantities of interest such as the average treatment effect by computing it for all observations, but then one must be aware that the quantity of interest may change during the matching procedure as some control units may be dropped.)

The imputation from the model can be done in at least two ways. Recall that the model is used to impute *the value that the outcome variable would take among the treated units if those treated units were actually controls*. Thus, one reasonable approach would be to fit a model to the matched data and create simulated predicted values of the dependent variable for the treated units with T_i switched counterfactually from 1 to 0. An alternative approach would be to fit a model without T by using only the outcomes of the matched control units (i.e., using only observations where $T_i = 0$). Then, given this fitted model, the missing outcomes $Y_i(0)$ are imputed for the matched treated units by using the values of the explanatory variables for the treated units. The first approach will usually have lower variance, since all observations are used, and the second may have less bias, since no assumption of constant parameters across the models of the potential outcomes under treatment and control is needed. See Ho, Imai, King, and Stuart (2007) for more details.

Other quantities of interest can also be computed at the parametric stage, following any procedures you would have followed in the absence of matching. The advantage is that if matching is done well your answers will be more robust to many small changes in parametric specification.

Chapter 3

User's Guide to MatchIt

3.1 Preprocessing via Matching

3.1.1 Quick Overview

The main command `matchit()` implements the matching procedures. A general syntax is:

```
> m.out <- matchit(treat ~ x1 + x2, data = mydata)
```

where `treat` is the dichotomous treatment variable, and `x1` and `x2` are pre-treatment covariates, all of which are contained in the data frame `mydata`. The dependent variable (or variables) may be included in `mydata` for convenience but is never used by MatchIt or included in the formula. This command creates the MatchIt object called `m.out`. Name the output object to see a quick summary of the results:

```
> m.out
```

3.1.2 Examples

To run any of the examples below, you first must load the library and and data:

```
> library(MatchIt)
> data(lalonde)
```

Our example data set is a subset of the job training program analyzed in Lalonde (1986) and Dehejia and Wahba (1999). MatchIt includes a subsample of the original data consisting of the National Supported Work Demonstration (NSW) treated group and the comparison sample from the Population Survey of Income Dynamics (PSID).[1] The variables in this data set include participation in the job training program (`treat`, which is equal to 1 if participated in the program, and 0 otherwise), age (`age`), years of education (`educ`), race

[1] This data set, `lalonde`, was created using NSWRE74_TREATED.TXT and CPS3_CONTROLS.TXT from http://www.columbia.edu/~rd247/nswdata.

(`black` which is equal to 1 if black, and 0 otherwise; `hispan` which is equal to 1 if hispanic, and 0 otherwise), marital status (`married`, which is equal to 1 if married, 0 otherwise), high school degree (`nodegree`, which is equal to 1 if no degree, 0 otherwise), 1974 real earnings (`re74`), 1975 real earnings (`re75`), and the main outcome variable, 1978 real earnings (`re78`).

3.1.2.1 Exact Matching

The simplest version of matching is exact. This technique matches *each* treated unit to *all* possible control units with exactly the same values on all the covariates, forming subclasses such that within each subclass all units (treatment and control) have the same covariate values. Exact matching is implemented in MatchIt using `method = "exact"`. Exact matching will be done on all covariates included on the right-hand side of the `formula` specified in the MatchIt call. There are no additional options for exact matching. (Exact restrictions on a subset of covariates can also be specified in nearest neighbor matching; see Section 3.1.2.3.) The following example can be run by typing `demo(exact)` at the R prompt,

```
> m.out <- matchit(treat ~ educ + black + hispan, data = lalonde,
                   method = "exact")
```

3.1.2.2 Subclassification

When there are many covariates (or some covariates can take a large number of values), finding sufficient exact matches will often be impossible. The goal of subclassification is to form subclasses, such that in each the distribution (rather than the exact values) of covariates for the treated and control groups are as similar as possible. Various subclassification schemes exist, including the one based on a scalar distance measure such as the propensity score estimated using the `distance` option (see Section 4.1.0.2.2). Subclassification is implemented in MatchIt using `method = "subclass"`.

The following example script can be run by typing `demo(subclass)` at the R prompt,

```
> m.out <- matchit(treat ~ re74 + re75 + educ + black + hispan + age,
                   data = lalonde, method = "subclass")
```

The above syntax forms 6 subclasses, which is the default number of subclasses, based on a distance measure (the propensity score) estimated using logistic regression. By default, each subclass will have approximately the same number of treated units.

Subclassification may also be used in conjunction with nearest neighbor matching described below, by leaving the default of `method = "nearest"` but adding the option `subclass`. When you choose this option, MatchIt selects matches using nearest neighbor matching, but after the nearest neighbor matches are chosen it places them into subclasses, and adds a variable to the output object indicating subclass membership.

3.1.2.3 Nearest Neighbor Matching

Nearest neighbor matching selects the r (default=1) best control matches for each individual in the treatment group (excluding those discarded using the `discard` option). Matching is done using a distance measure specified by the `distance` option (default=logit). Matches are chosen for each treated unit one at a time, with the order specified by the `m.order` command (default=largest to smallest). At each matching step we choose the control unit that is not yet matched but is closest to the treated unit on the distance measure.

Nearest neighbor matching is implemented in MatchIt using the `method = "nearest"` option. The following example script can be run by typing `demo(nearest)`:

```
> m.out <- matchit(treat ~ re74 + re75 + educ + black + hispan + age,
                   data = lalonde, method = "nearest")
```

3.1.2.4 Optimal Matching

The default nearest neighbor matching method in MatchIt is "greedy" matching, where the closest control match for each treated unit is chosen one at a time, without trying to minimize a global distance measure. In contrast, "optimal" matching finds the matched samples with the smallest average absolute distance across all the matched pairs. Gu and Rosenbaum (1993) find that greedy and optimal matching approaches generally choose the same sets of controls for the overall matched samples, but optimal matching does a better job of minimizing the distance within each pair. In addition, optimal matching can be helpful when there are not many appropriate control matches for the treated units.

Optimal matching is performed with MatchIt by setting `method = "optimal"`, which automatically loads an add-on package called `optmatch` (Hansen 2004). The following example can also be run by typing `demo(optimal)` at the R prompt. We conduct 2:1 optimal ratio matching based on the propensity score from the logistic regression.

```
> m.out <- matchit(treat ~ re74 + re75 + age + educ, data = lalonde,
                   method = "optimal", ratio = 2)
```

3.1.2.5 Full Matching

Full matching is a particular type of subclassification that forms the subclasses in an optimal way (Rosenbaum 2002; Hansen 2004). A fully matched sample is composed of matched sets, where each matched set contains one treated unit and one or more controls (or one control unit and one or more treated units). As with subclassification, the only units not placed into a subclass will be those discarded (if a `discard` option is specified) because they are outside the range of common support. Full matching is optimal in terms of minimizing a weighted average of the estimated distance measure between each treated subject and each control subject within each subclass.

Full matching can be performed with MatchIt by setting `method = "full"`. Just as with optimal matching, we use the `optmatch` package (Hansen 2004), which automatically

loads when needed. The following example with full matching (using the default propensity score based on logistic regression) can also be run by typing `demo(full)` at the R prompt:

```
> m.out <- matchit(treat ~ age + educ + black + hispan + married +
                   nodegree + re74 + re75, data = lalonde, method = "full")
```

3.1.2.6 Genetic Matching

Genetic matching automates the process of finding a good matching solution (Diamond and Sekhon 2005). The idea is to use a genetic search algorithm to find a set of weights for each covariate such that the a version of optimal balance is achieved after matching. As currently implemented, matching is done with replacement using the matching method of Abadie and Imbens (2007) and balance is determined by two univariate tests, paired t-tests for dichotomous variables and a Kolmogorov-Smirnov test for multinomial and continuous variables, but these options can be changed.

Genetic matching can be performed with MatchIt by setting `method = "genetic"`, which automatically loads the `Matching` (**?**) package. The following example of genetic matching (using the estimated propensity score based on logistic regression as one of the covariates) can also be run by typing `demo(genetic)`:

```
> m.out <- matchit(treat ~ age + educ + black + hispan + married + nodegree +
                   re74 + re75, data = lalonde, method = "genetic")
```

3.1.2.7 Coarsened Exact Matching

Coarsened Exact Matching (CEM) is a Monotonoic Imbalance Bounding (MIB) matching method — which means that the balance between the treated and control groups is chosen by the user ex ante rather than discovered through the usual laborious process of checking after the fact and repeatedly reestimating, and so that adjusting the imbalance on one variable has no effect on the maximum imbalance of any other. CEM also strictly bounds through ex ante user choice both the degree of model dependence and the average treatment effect estimation error, eliminates the need for a separate procedure to restrict data to common empirical support, meets the congruence principle, is robust to measurement error, works well with multiple imputation methods for missing data, and is extremely fast computationally even with very large data sets. CEM also works well for multicategory treatments, determining blocks in experimental designs, and evaluating extreme counterfactuals (**?**).

CEM can be performed with MatchIt by setting `method = "cem"`, which automatically loads the `cem` package. The following examples of CEM (with automatic coarsening) can also be run by typing `demo(cem)`:

```
m.out <- matchit(treat ~ age + educ + black + hispan + married + nodegree
                 + re74 + re75, data = lalonde, method = "cem")
```

3.2 Checking Balance

3.2.1 Quick Overview

To check balance, use `summary(m.out)` for numerical summaries and `plot(m.out)` for graphical summaries.

3.2.2 Details

3.2.2.1 The `summary()` Command

The `summary()` command gives measures of the balance between the treated and control groups in the full (original) data set, and then in the matched data set. If the matching worked well, the measures of balance should be smaller in the matched data set (smaller values of the measures indicate better balance).

The `summary()` output for subclassification is the same as that for other types of matching, except that the balance statistics are shown separately for each subclass, and the overall balance in the matched samples is calculated by aggregating across the subclasses, where each subclass is weighted by the number of units in the subclass. For exact matching, the covariate values within each subclass are guaranteed to be the same, and so the measures of balance are not output for exact matching; only the sample sizes in each subclass are shown.

- **Balance statistics:** The statistics the `summary()` command provides include means, the original control group standard deviation (where applicable), mean differences, standardized mean differences, and (median, mean and maximum) Quantile-Quantile (Q-Q) plot differences. In addition, the `summary()` command will report (a) the matched call, (b) how many units were matched, unmatched, or discarded due to the `discard` option (described below), and (c) the percent improvement in balance for each of the balance measures, defined as $100((|a| - |b|)/|a|)$, where a is the balance before and b is the balance after matching. For each set of units (original and matched data sets, with weights used as appropriate in the matched data sets), the following statistics are provided:

    1. "Means Treated" and "Means Control" show the weighted means in the treated and control groups
    2. "SD Control" is the standard deviation calculated in the control group (where applicable)
    3. "Mean Diff" is the difference in means between the groups
    4. The final three columns of the summary output give summary statistics of a Q-Q plot (see below for more information on these plots). Those columns give the median, mean, and maximum distance between the two empirical quantile functions (treated and control groups). Values greater than 0 indicate deviations between the groups in some part of the empirical distributions. The plots of the

two empirical quantile functions themselves, described below, can provide further insight into which part of the covariate distribution has differences between the two groups.

- **Additional options:** Three options to the `summary()` command can also help with assessing balance and respecifying the propensity score model, as necessary. First, the `interactions = TRUE` option with `summary()` shows the balance of all squares and interactions of the covariates used in the matching procedure. Large differences in higher order interactions usually are a good indication that the propensity score model (the distance measure) needs to be respecified. Similarly, the `addlvariables` option with `summary()` will provide balance measures on additional variables not included in the original matching procedure. If a variable (or interaction of variables) not included in the original propensity score model has large imbalances in the matched groups, including that variable in the next model specification may improve the resulting balance on that variable. Because the outcome variable is not used in the matching procedure, a variety of matching methods can be tried, and the one that leads to the best resulting balance chosen. Finally, the `standardize = TRUE` option will print out standardized versions of the balance measures, where the mean difference is standardized (divided) by the standard deviation in the original treated group.

3.2.2.2 The `plot()` Command

We can also examine the balance graphically using the `plot()` command, which provides three types of plots: jitter plots of the distance measure, Q-Q plots of each covariate, and histograms of the distance measure. For subclassification, separate Q-Q plots can be printed for each subclass. The jitter plot for subclassification is the same as that for other types of matching, with the addition of vertical lines indicating the subclass cut-points. With the histogram option, 4 histograms are provided: the original treated and control groups and the matched treated and control groups. For the Q-Q plots and the histograms, the weights that result after matching are used to create the plots.

Three examples of the output from the `plot()` command are shown in Figure 3.1. If the empirical distributions are the same in the treated and control groups, the points in the Q-Q plots would all lie on the 45 degree line (lower left panel of Figure 3.1). Deviations from the 45 degree line indicate differences in the empirical distribution. The jitter plot (top panel) shows the overall distribution of propensity scores in the treated and control groups. In the jitter plot, which can be created by setting `type = "jitter"`, the size of each point is proportional to the weight given to that unit. Observation names can be interactively identified by clicking the first mouse button near the units. The histograms (lower right panel) can be plotted by setting `type = "hist"`.

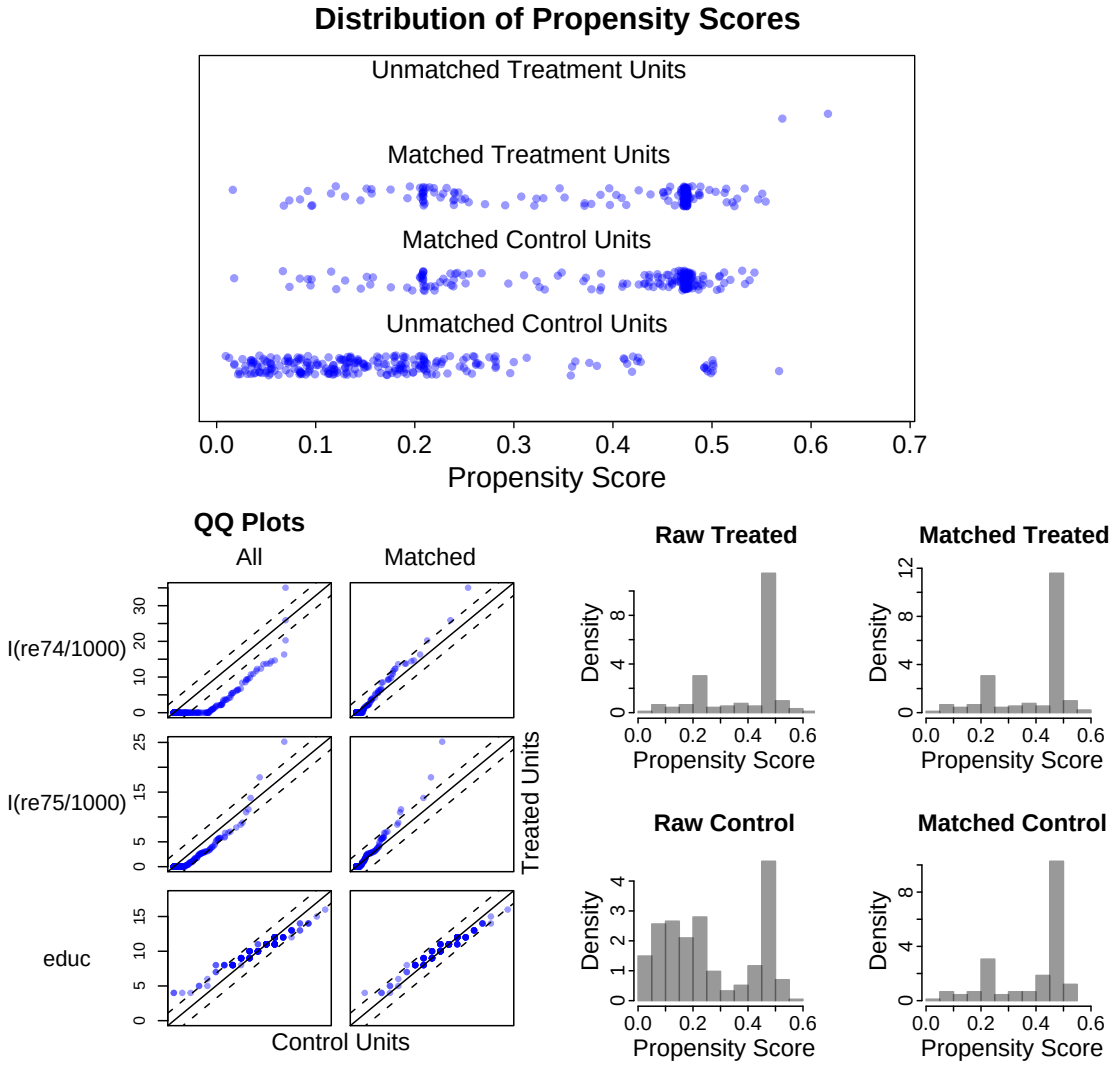

Figure 3.1: Examples of the three types of output from the `plot` command resulting from matching on the /textttlalonde data set based on real earnings in 1974 (`re74`) divided by 1000, real earnings in 1975 (`re75`) divided by 1000, years of education (`educ`), Hispanic (`hispan`) and marital status (`married`). Observations in both the treated and the control groups outside the support of the distance measure were discarded. The upper plot shows the jitter plot of the distance measure. The lower left plot shows the QQ plots for the first three covariates (`I(re74/1000)`, `I(re75/1000)`, `educ`). The lower right plot shows the histograms of the density of propensity scores for observations before and after matching.

3.3 Conducting Analyses after Matching

Any software package may be used for parametric analysis following MatchIt. This includes any of the relevant R packages, or other statistical software by exporting the resulting matched data sets using R commands such as `write.csv()` and `write.table()` for ASCII files or `write.dta()` in the `foreign` package for a STATA binary file.

When variable numbers of treated and control units have been matched to each other (e.g., through exact matching, full matching, or k:1 matching with replacement), the weights created by MatchIt should be used (e.g., in a weighted regression) to ensure that the matched treated and control groups are weighted up to be similar. Users should also remember that the weights created by MatchIt estimate the average treatment effect on the treated, with the control units weighted to resemble the treated units. See below for more detail on the weights. With subclassification, estimates should be obtained within each subclass and then aggregated across subclasses. When it is not possible to calculate an effect within each subclass, again the weights can be used to weight the matched units.

In this section, we show how to use Zelig with MatchIt. Zelig (Imai et al. 2006) is an R package that implements a large variety of statistical models (using numerous existing R packages) with a single easy-to-use interface, gives easily interpretable results by simulating quantities of interest, provides numerical and graphical summaries, and is easily extensible to include new methods.

3.3.1 Quick Overview

The general syntax is as follows. First, we use `match.data()` to create the matched data from the MatchIt output object (`m.out`) by excluding unmatched units from the original data, and including information produced by the particular matching procedure (i.e., primarily a new data set, but also information that may result such as weights, subclasses, or the distance measure).

```
> m.data <- match.data(m.out)
```

where `m.data` is the resulting matched data. Zelig analyses all use three commands — `zelig`, `setx`, and `sim`. For example, the basic statistical analysis is performed first:

```
> z.out <- zelig(Y ~ treat + x1 + x2, model = mymodel, data = m.data)
```

where `Y` is the outcome variable, `mymodel` is the selected model, and `z.out` is the output object from `zelig`. This output object includes estimated coefficients, standard errors, and other typical outputs from your chosen statistical model. Its contents can be examined via `summary(z.out)` or `plot(z.out)`, but the idea of Zelig is that these statistical results are typically only intermediate quantities needed to compute your ultimate quantities of interest, which in the case of matching are usually causal inferences. To get these causal quantities, we use Zelig's other two commands. Thus, we can set the explanatory variables at their means (the default) and change the treatment variable from a 0 to a 1:

```
> x.out <- setx(z.out, treat=0)
> x1.out <- setx(z.out, treat=1)
```

and finally compute the resulting estimates of the causal effects and examine a summary:

```
> s.out <- sim(z.out, x = x.out, x1 = x1.out)
> summary(s.out)
```

3.3.2 Examples

We now give four examples using the Lalonde data. They are meant to be read sequentially. You can run these example commands by typing `demo(analysis)`. Although we use the linear least squares model in these examples, a wide range of other models are available in Zelig (for the list of supported models, see `http://gking.harvard.edu/zelig/docs/Models_Zelig_Can.html`.

To load the Zelig package after installing it, type

```
> library(Zelig)
```

Model-Based Estimates In our first example, we conduct a standard parametric analysis and compute quantities of interest in the most common way. We begin with nearest neighbor matching with a logistic regression-based propensity score, discarding control units outside the convex hull of the treated units (King and Zeng 2006, 2007):

```
> m.out <- matchit(treat ~ age + educ + black + hispan + nodegree +
                     married + re74 + re75, method = "nearest", discard
                     = "hull.control", data = lalonde)
```

Then we check balance using the summary and plot procedures (which we don't show here). When the best balance is achieved, we run the parametric analysis:

```
> z.out <- zelig(re78 ~ treat + age + educ + black + hispan + nodegree +
                   married + re74 + re75, data = match.data(m.out),
                   model = "ls")
```

and then set the explanatory variables at their means (the default) and change the treatment variable from a 0 to a 1:

```
> x.out <- setx(z.out, treat=0)
> x1.out <- setx(z.out, treat=1)
```

and finally compute the result and examine a summary:

```
> s.out <- sim(z.out, x = x.out, x1 = x1.out)
> summary(s.out)
```

Average Treatment Effect on the Treated We illustrate now how to estimate the average treatment effect on the treated in a way that is quite robust. We do this by estimating the coefficients in the control group alone.

We begin by conducting nearest neighbor matching with a logistic regression-based propensity score:

```
> m.out1 <- matchit(treat ~ age + educ + black + hispan + nodegree +
                    married + re74 + re75, method = "nearest", data = lalonde)
```

Then we check balance using the summary and plot procedures (which we don't show here). We reestimate the matching procedure until we achieve the best balance possible. (The running examples here are meant merely to illustrate, not to suggest that we've achieved the best balance.) Then we go to Zelig, and in this case choose to fit a linear least squares model to the control group only:

```
> z.out1 <- zelig(re78 ~ age + educ + black + hispan + nodegree +
                  married + re74 + re75, data = match.data(m.out1, "control"),
                  model = "ls")
```

where the `"control"` option in `match.data()` extracts only the matched control units and `ls` specifies least squares regression. In a smaller data set, this example should probably be changed to include all the data in this estimation (using `data = match.data(m.out1)` for the data) and by including the treatment indicator (which is excluded in the example since its a constant in the control group.) Next, we use the coefficients estimated in this way from the control group, and combine them with the values of the covariates set to the values of the treated units. We do this by choosing conditional prediction (which means use the observed values) in `setx()`. The `sim()` command does the imputation.

```
> x.out1 <- setx(z.out1, data = match.data(m.out1, "treat"), cond = TRUE)
> s.out1 <- sim(z.out1, x = x.out1)
```

Finally, we obtain a summary of the results by

```
> summary(s.out1)
```

Average Treatment Effect (Overall) To estimate the average treatment effect, we continue with the previous example and fit the linear model to the *treatment group*:

```
> z.out2 <- zelig(re78 ~ age + educ + black + hispan + nodegree +
                  married + re74 + re75, data = match.data(m.out1, "treat"),
                  model = "ls")
```

We then conduct the same simulation procedure in order to impute the counterfactual outcome for the *control group*,

```
> x.out2 <- setx(z.out2, data = match.data(m.out1, "control"), cond = TRUE)
> s.out2 <- sim(z.out2, x = x.out2)
```

In this calculation, Zelig is computing the difference between observed and the expected values. This means that the treatment effect for the control units is the effect of control (observed control outcome minus the imputed outcome under treatment from the model). Hence, to combine treatment effects just reverse the signs of the estimated treatment effect of controls.

```
> ate.all <- c(s.out1$qi$att.ev, -s.out2$qi$att.ev)
```

The point estimate, its standard error, and the 95% confidence interval is given by

```
> mean(ate.all)
> sd(ate.all)
> quantile(ate.all, c(0.025, 0.975))
```

Subclassification In subclassification, the average treatment effect estimates are obtained separately for each subclass, and then aggregated for an overall estimate. Estimating the treatment effects separately for each subclass, and then aggregating across subclasses, can increase the robustness of the ultimate results since the parametric analysis within each subclass requires only local rather than global assumptions. However, fewer observations are obviously available within each subclass, and so this option is normally chosen for larger data sets.

We begin this example by conducting subclassification with four subclasses,

```
> m.out2 <- matchit(treat ~ age + educ + black + hispan + nodegree +
                    married + re74 + re75, data = lalonde,
                    method = "subclass", subclass = 4)
```

When balance is as good as we can get it, we then fit a linear regression within each subclass by controlling for the estimated propensity score (called `distance`) and other covariates. In most software, this would involve running four separate regressions and then combining the results. In Zelig, however, all we need to do is to use the `by` option:

```
> z.out3 <- zelig(re78 ~ re74 + re75 + distance,
                  data = match.data(m.out2, "control"),
                  model = "ls", by = "subclass")
```

The same set of commands as in the first example are used to do the imputation of the counterfactual outcomes for the treated units:

```
> x.out3 <- setx(z.out3, data = match.data(m.out2, "treat"), fn = NULL,
                 cond = TRUE)
> s.out3 <- sim(z.out3, x = x.out3)
> summary(s.out3)
```

It is also possible to get the summary result for each subclass. For example, the following command summarizes the result for the second subclass.

```
> summary(s.out3, subset = 2)
```

How Adjustment After Exact Matching Has No Effect Regression adjustment after exact one-to-one exact matching gives the identical answer as a simple, unadjusted difference in means. General exact matching, as implemented in MatchIt, allows one-to-many matches, so to see the same result we must weight when adjusting. In other words: weighted regression adjustment after general exact matching gives the identical answer as a simple, unadjusted weighted difference in means. For example:

```
> m.out <- matchit(treat ~ educ + black + hispan, data = lalonde,
                   method = "exact")
> m.data <- match.data(m.out)
> ## weighted diff in means
> weighted.mean(mdata$re78[mdata$treat == 1], mdata$weights[mdata$treat==1])
        - weighted.mean(mdata$re78[mdata$treat==0], mdata$weights[mdata$treat==0])
[1] 807
> ## weighted least squares without covariates
> zelig(re78 ~ treat, data = m.data, model = "ls", weights = "weights")

Call:
zelig(formula = re78 ~ treat, model = "ls", data = m.data, weights =
"weights")

Coefficients:
(Intercept)        treat
       5524          807

> ## weighted least squares with covariates
> zelig(re78 ~ treat + black + hispan + educ, data = m.data, model = "ls",
        weights = "weights")

Call:
```

```
zelig(formula = re78 ~ treat + black + hispan + educ, model = "ls",
data = m.data, weights = "weights")

Coefficients:
(Intercept)        treat        black       hispan         educ
        314          807        -1882          258          657
```

Chapter 4

Reference Manual

4.1 `matchit()`: Implementation of Matching Methods

Use `matchit()` to implement a variety of matching procedures including exact matching, nearest neighbor matching, subclassification, optimal matching, genetic matching, and full matching. The output of `matchit()` can be analyzed via any standard R package, by exporting the data for use in another program, or most simply via Zelig in R.

4.1.0.1 Syntax

```
> m.out <- matchit(formula, data, method = "nearest", verbose = FALSE, ...)
```

4.1.0.2 Arguments

4.1.0.2.1 Arguments for All Matching Methods

- `formula`: formula used to calculate the distance measure for matching (e.g., the propensity score model). It takes the usual syntax of R formulas, `treat ~ x1 + x2`, where `treat` is a binary treatment indicator, and `x1` and `x2` are the pre-treatment covariates. Both the treatment indicator and pre-treatment covariates must be contained in the same data frame, which is specified as `data` (see below). All of the usual R syntax for formulas work here. For example, `x1:x2` represents the first order interaction term between `x1` and `x2`, and `I(x1 ^ 2)` represents the square term of `x1`. See `help(formula)` for details.
- `data`: the data frame containing the variables called in `formula`.
- `method`: the matching method (default = `"nearest"`, nearest neighbor matching). Currently, `"exact"` (exact matching), `"full"` (full matching), `"nearest"` (nearest neighbor matching), `"optimal"` (optimal matching), `"subclass"` (subclassification), `"genetic"` (genetic matching), and `"cem"` (coarsened exact matching) are available. Note that within each of these matching methods, MatchIt offers a variety of options. See below for more details.

- `verbose`: a logical value indicating whether to print the status of the matching algorithm (default = `FALSE`).

4.1.0.2.2 Additional Arguments for Specification of Distance Measures The following arguments specify distance measures that are used for matching methods. These arguments apply to all matching methods *except exact matching*.

- `distance`: the method used to estimate the distance measure (default = `"logit"`, logistic regression) or a numerical vector of user's own distance measure. Before using any of these techniques, it is best to understand the theoretical groundings of these techniques and to evaluate the results. Most of these methods (such as logistic or probit regression) define the distance by first estimating the propensity score, defined as the probability of receiving treatment, conditional on the covariates. Available methods include:
  - `"mahalanobis"`: the Mahalanobis distance measure.
  - binomial generalized linear models with one of the following link functions:
    - `"logit"`: logistic link
    - `"linear.logit"`: logistic link with linear propensity score[1]
    - `"probit"`: probit link
    - `"linear.probit"`: probit link with linear propensity score
    - `"cloglog"`: complementary log-log link
    - `"linear.cloglog"`: complementary log-log link with linear propensity score
    - `"log"`: log link
    - `"linear.log"`: log link with linear propensity score
    - `"cauchit"` Cauchy CDF link
    - `"linear.cauchit"` Cauchy CDF link with linear propensity score
  - Choose one of the following generalized additive models (see `help(gam)` for more options).
    - `"GAMlogit"`: logistic link
    - `"GAMlinear.logit"`: logistic link with linear propensity score
    - `"GAMprobit"`: probit link
    - `"GAMlinear.probit"`: probit link with linear propensity score
    - `"GAMcloglog"`: complementary log-log link
    - `"GAMlinear.cloglog"`: complementary log-log link with linear propensity score
    - `"GAMlog"`: log link
    - `"GAMlinear.log"`: log link with linear propensity score,

[1]The linear propensity scores are obtained by transforming back onto a linear scale.

    * "GAMcauchit": Cauchy CDF link
    * "GAMlinear.cauchit": Cauchy CDF link with linear propensity score
  - "nnet": neural network model. See help(nnet) for more options.
  - "rpart": classification trees. See help(rpart) for more options.

- distance.options: optional arguments for estimating the distance measure. The input to this argument should be a list. For example, if the distance measure is estimated with a logistic regression, users can increase the maximum IWLS iterations by distance.options = list(maxit = 5000). Find additional options for general linear models using help(glm) or help(family), for general additive models using help(gam), for neutral network models help(nnet), and for classification trees help(rpart).

- discard: specifies whether to discard units that fall outside some measure of support of the distance measure (default = "none", discard no units). Discarding units may change the quantity of interest being estimated by changing the observations left in the analysis. Enter a logical vector indicating which unit should be discarded or choose from the following options:

  - "none": no units will be discarded before matching. Use this option when the units to be matched are substantially similar, such as in the case of matching treatment and control units from a field experiment that was close to (but not fully) randomized (e.g., Imai 2005), when caliper matching will restrict the donor pool, or when you do not wish to change the quantity of interest and the parametric methods to be used post-matching can be trusted to extrapolate.
  - "hull.both": all units that are not within the convex hull will be discarded. See King and Zeng (2006, 2007) for information about the convex hull in this context and as a measure of model dependence.
  - "both": all units (treated and control) that are outside the support of the distance measure will be discarded.
  - "hull.control": only control units that are not within the convex hull of the treated units will be discarded.
  - "control": only control units outside the support of the distance measure of the treated units will be discarded. Use this option when the average treatment effect on the treated is of most interest and when you are unwilling to discard non-overlapping treatment units (which would change the quantity of interest).
  - "hull.treat": only treated units that are not within the convex hull of the control units will be discarded.
  - "treat": only treated units outside the support of the distance measure of the control units will be discarded. Use this option when the average treatment effect on the control units is of most interest and when unwilling to discard control units.

- `reestimate`: If `FALSE` (default), the model for the distance measure will not be re-estimated after units are discarded. The input must be a logical value. Re-estimation may be desirable for efficiency reasons, especially if many units were discarded and so the post-discard samples are quite different from the original samples.

4.1.0.2.3 Additional Arguments for Subclassification

- `sub.by`: criteria for subclassification. Choose from: `"treat"` (default), the number of treatment units; `"control"`, the number of control units; or `"all"`, the total number of units. Changing the default will likely also signal a change in your quantity of interest from the average treatment effect on the treated to other quantities.

- `subclass`: either a scalar specifying the number of subclasses, or a vector of probabilities bounded between 0 and 1, which create quantiles of the distance measure using the units in the group specified by `sub.by` (default = `subclass = 6`).

4.1.0.2.4 Additional Arguments for Nearest Neighbor Matching

- `m.order`: the order in which to match treatment units with control units.
  - `"largest"` (default): matches from the largest value of the distance measure to the smallest.
  - `"smallest"`: matches from the smallest value of the distance measure to the largest.
  - `"random"`: matches in random order.

- `replace`: logical value indicating whether each control unit can be matched to more than one treated unit (default = `replace = FALSE`, each control unit is used at most once – i.e., sampling without replacement). For matching with replacement, use `replace = TRUE`. After matching with replacement, the weights can be used to reflect the frequency with which each control unit was matched.

- `ratio`: the number of control units to match to each treated unit (default = `1`). If matching is done without replacement and there are fewer control units than `ratio` times the number of eligible treated units (i.e., there are not enough control units for the specified method), then the higher ratios will have `NA` in place of the matching unit number in `match.matrix`.

- `exact`: variables on which to perform exact matching within the nearest neighbor matching (default = `NULL`, no exact matching). If `exact` is specified, only matches that exactly match on the covariates in `exact` will be allowed. Within the matches that match on the variables in `exact`, the match with the closest distance measure will be chosen. `exact` should be entered as a vector of variable names (e.g., `exact = c("X1", "X2")`).

- `caliper`: the number of standard deviations of the distance measure within which to draw control units (default = `0`, no caliper matching). If a caliper is specified, a control unit within the caliper for a treated unit is randomly selected as the match for that treated unit. If `caliper != 0`, there are two additional options:
  - `calclosest`: whether to take the nearest available match if no matches are available within the `caliper` (default = `FALSE`).
  - `mahvars`: variables on which to perform Mahalanobis-metric matching within each caliper (default = `NULL`). Variables should be entered as a vector of variable names (e.g., `mahvars = c("X1", "X2")`). If `mahvars` is specified without `caliper`, the caliper is set to 0.25.
- `subclass` and `sub.by`: See the options for subclassification for more details on these options. If a `subclass` is specified within `method = "nearest"`, the matched units will be placed into subclasses after the nearest neighbor matching is completed.

4.1.0.2.5 Additional Arguments for Optimal Matching

- `ratio`: the number of control units to be matched to each treatment unit (default = `1`).
- `...`: additional inputs that can be passed to the `fullmatch()` function in the `optmatch` package. See `help(fullmatch)` or http://www.stat.lsa.umich.edu/~bbh/optmatch.html for details.

4.1.0.2.6 Additional Arguments for Full Matching See `help(fullmatch)` (part of this information is copied below) or http://www.stat.lsa.umich.edu/~bbh/optmatch.html for details.

- `min.controls`: The minimum ratio of controls to treatments that is to be permitted within a matched set: should be nonnegative and finite. If `min.controls` is not a whole number, the reciprocal of a whole number, or zero, then it is rounded down to the nearest whole number or reciprocal of a whole number.
- `max.controls`: The maximum ratio of controls to treatments that is to be permitted within a matched set: should be positive and numeric. If `max.controls` is not a whole number, the reciprocal of a whole number, or `Inf`, then it is rounded up to the nearest whole number or reciprocal of a whole number.
- `omit.fraction`: Optionally, specify what fraction of controls or treated subjects are to be rejected. If `omit.fraction` is a positive fraction less than one, then `fullmatch()` leaves up to that fraction of the control reservoir unmatched. If `omit.fraction` is a negative number greater than -1, then `fullmatch()` leaves up to |omit.fraction| of the treated group unmatched. Positive values are only accepted if max.controls >= 1;

negative values, only if min.controls <= 1. If `omit.fraction` is not specified, then only those treated and control subjects without permissible matches among the control and treated subjects, respectively, are omitted.

- `...`: Additional inputs that can be passed to the `fullmatch()` function in the `optmatch` package.

4.1.0.2.7 Additional Arguments for Genetic Matching The available options are listed below.

- `ratio`: the number of control units to be matched to each treatment unit (default = `1`).
- `...`: additional minor inputs that can be passed to the `GenMatch()` function in the `Matching` package. See `help(GenMatch)` or http://sekhon.polisci.berkeley.edu/library/Matching/html/GenMatch.html for details.

4.1.0.2.8 Additional Arguments for Coarsened Exact Matching The available options are listed here:

- cutpoints named list each describing the cutpoints for the variables. Each list element is either a vector of cutpoints, a number of cutpoints, a method for automatic bin contruction.
- k2k return k-to-k matching?
- verbose controls level of verbosity
- dist user defined distance function
- `...`: additional minor inputs that can be passed to the `cem()` function in the `cem` package. See `help(cem)` or http://gking.harvard.edu/cem for details.

4.1.0.3 Output Values

Regardless of the type of matching performed, the `matchit` output object contains the following elements:[2]

- `call`: the original `matchit()` call.
- `formula`: the formula used to specify the model for estimating the distance measure.
- `model`: the output of the model used to estimate the distance measure. `summary(m.out$model)` will give the summary of the model where `m.out` is the output object from `matchit()`.

[2]When inapplicable or unnecessary, these elements may equal `NULL`. For example, when exact matching, `match.matrix = NULL`.

- `match.matrix`: an $n_1 \times$ `ratio` matrix where:
  - the row names represent the names of the treatment units (which match the row names of the data frame specified in `data`).
  - each column stores the name(s) of the control unit(s) matched to the treatment unit of that row. For example, when the `ratio` input for nearest neighbor or optimal matching is specified as 3, the three columns of `match.matrix` represent the three control units matched to one treatment unit).
  - `NA` indicates that the treatment unit was not matched.
- `discarded`: a vector of length n that displays whether the units were ineligible for matching due to common support restrictions. It equals `TRUE` if unit i was discarded, and it is set to `FALSE` otherwise.
- `distance`: a vector of length n with the estimated distance measure for each unit.
- `weights`: a vector of length n with the weights assigned to each unit in the matching process. Unmatched units have weights equal to 0. Matched treated units have weight 1. Each matched control unit has weight proportional to the number of treatment units to which it was matched, and the sum of the control weights is equal to the number of uniquely matched control units.
- `subclass`: the subclass index in an ordinal scale from 1 to the total number of subclasses as specified in `subclass` (or the total number of subclasses from full or exact matching). Unmatched units have `NA`.
- `q.cut`: the subclass cut-points that classify the distance measure.
- `treat`: the treatment indicator from `data` (the left-hand side of `formula`).
- `X`: the covariates used for estimating the distance measure (the right-hand side of `formula`). When applicable, `X` is augmented by covariates contained in `mahvars` and `exact`.
- `nn`: A basic summary table of matched data (e.g., the number of matched units)

4.2 `summary()`: Numerical Summaries of Balance

The `summary()` command returns numerical summaries of balance diagnostics.

4.2.0.1 Syntax

```
summary(object, interactions = FALSE, addlvariables = NULL, standardize = FALSE, ...)
```

4.2.0.2 Arguments

- `object`: the output from `matchit()`.
- `interactions`: an option to calculate summary statistics in `sum.all` and `sum.matched` for all covariates, their squares, and two-way interactions when `interactions = TRUE` and only the covariates themselves when `interactions = FALSE`, (DEFAULT = `FALSE`).
- `addlvariables`: additional variables on which to calculate the diagnostic statistics (in addition to the variables included in the matching procedure) (DEFAULT = `NULL`). `addlvariables`: a data frame containing additional variables whose balance is examined. The data should come with the same number of units and units in the same order as in the data set used for `matchit()`.
- `standardize`: a logical variable indicating whether to standardize balance measures, i.e., whether the difference in means should be divided by the standard deviation in the original treated group. (DEFAULT = `FALSE`)

4.2.0.3 Output Values

The output from the `summary()` command includes the following elements, when applicable:

- The original assignment model call.
- `sum.all`: a data frame that contains variable names and interactions down the row names, and summary statistics on *all observations* in each of the columns. The columns in `sum.all` contain:
  - means of all covariates X for treated and control units, where `Means Treated`= $\mu_{X|T=1} = \frac{1}{n_1}\sum_{T=1} X_i$ and `Means Control`= $\mu_{X|T=0} = \frac{1}{n_0}\sum_{T=0} X_i$,
  - standard deviation in the control group for all covariates X, where applicable,

$$s_{x|T=0} = \sqrt{\frac{\sum_{i\in\{i:T_i=0\}}(X_i - \mu_{X|T=0})^2}{n_0 - 1}}.$$

  - balance statistics of the original data (before matching), which compare treated and control covariate distributions. If `standardize = FALSE`, balance measures will be presented on the original scale. Specifically, mean differences (`Mean Diff.`) as well as the median, mean, and maximum value of differences in empirical quantile functions for each covariate will be given (`eQQ Med`, `eQQ Mean`, and `eQQ Max`, respectively). If `standardize = TRUE`, the balance measures will be standardized. Standardized mean differences (`Std. Mean Diff.`), defined as $\frac{\mu_{X|T=1}-\mu_{X|T=0}}{s_{x|T=1}}$, as well as the median, mean, and maximum value of differences in empirical cumulative distribution functions for each covariate will be given (`eCDF Med`, `eCDF Mean`, and `eCDF Max`, respectively).

- `sum.matched`: a data frame which contains variable names down the row names, and summary statistics on only the *matched observations* in each of the columns. Specifically, the columns in `sum.matched` contain the following elements:
  - weighted means for matched treatment units and matched control units of all covariates X and their interactions, where `Means Treated`= $\mu_{wX|T=1} = \frac{1}{n_1}\sum_{T=1} w_i X_i$ and `Means Control`= $\mu_{wX|T=0} = \frac{1}{n_0}\sum_{T=0} w_i X_i$,
  - weighted standard deviations in the matched control group for all covariates X, where applicable, where `SD` = $s_{wX} = \sqrt{\frac{1}{n}\sum_i (w_i X_i - \overline{X}^*)^2}$, where $\overline{X}^*$ is the weighted mean of X in the matched control group, and
  - balance statistics of the matched data (after matching), which compare treated and control covariate distributions. If `standardize = FALSE`, balance measures will be presented on the original scale. Specifically, mean differences (`Mean Diff.`) as well as the median, mean, and maximum value of differences in empirical quantile functions for each covariate will be given (`eQQ Med`, `eQQ Mean`, and `eQQ Max`, respectively). If `standardize = TRUE`, the balance measures will be standardized. Standardized mean differences (`Std. Mean Diff.`), defined as $\frac{\mu_{wX|T=1} - \mu_{wX|T=0}}{s_{x|T=1}}$, as well as the median, mean, and maximum value of differences in empirical cumulative distribution functions for each covariate will be given (`eCDF Med`, `eCDF Mean`, and `eCDF Max`, respectively).

  where w represents the vector of `weights`.

- `reduction`: the percent reduction in the difference in means achieved in each of the balance measures in `sum.all` and `sum.matched`, defined as $100(|a| - |b|)/|a|$, where a was the value of the balance measure before matching and b is the value of the balance measure after matching.

- `nn`: the sample sizes in the full and matched samples and the number of discarded units, by treatment and control.

- `q.table`: an array that contains the same information as `sum.matched` by subclass.

- `qn`: the sample sizes in the full and matched samples and the number of discarded units, by subclass and by treatment and control.

- `match.matrix`: the same object is contained in the output of `matchit()`.

4.3 `plot()`: Graphical Summaries of Balance

4.3.1 Plot options for the matchit object

The `plot()` command allows you to check the distributions of propensity scores and covariates in the assignment model, squares, and interactions, and within each subclasses if

specified.

4.3.1.1 Syntax

```
> plot(m.out, discrete.cutoff = 5, type = "QQ",
       numdraws = 5000, interactive = TRUE, which.xs = NULL, ...)
```

4.3.1.2 Arguments

- `type`: type of output graph. `type = "QQ"` (default) outputs empirical quantile-quantile plots of each covariate to check balance of marginal distributions. Alternatively, `type = "jitter"` outputs jitter plots of the propensity score for treated and control units. Finally, `type="hist"` outputs histograms of the propensity score in the original treated and control groups and weighted histograms of the propensity score in the matched treated and control groups.

- `discrete.cutoff`: For quantile-quantile plots, discrete covariates that take 5 or fewer values are jittered for visibility. This may be changed by setting this argument to any other positive integer.

- `interactive`: If `TRUE` (default), users can identify individual units by clicking on the graph with the left mouse button, and (when applicable) choose subclasses to plot.

- `which.xs`: For quantitle-quantile plots, specifies particular covariate names in a character vector to plot only a subset of the covariates.

- `subclass`: If `interactive = FALSE`, users can specify which subclass to plot.

4.3.1.3 Output Values

- Empirical quantile-quantile plot: This graph plots covariate values that fall in (approximately) the same quantile of treated and control distributions. Control unit quantile values are plotted on the x-axis, and treated unit quantile values are plotted on the y-axis. If values fall below the 45 degree line, control units generally take lower values of the covariate. Data points that fall exactly on the 45 degree line indicate that the marginal distributions are identical.

- Jitter plots: This graph plots jittered estimated propensity scores of treated and control units. Dark diamonds indicate matched units and grey diamonds indicate unmatched or discarded units. The area of the diamond is proportional to the weights. Vertical lines are plotted if subclassification is used.

- Histograms: This graph plots histograms of the estimated propensity scores in the original treated and control groups and weighted histograms of the estimated propensity scores in the matched treated and control groups. Plots can be compared vertically to quickly check the balance before and after matching.

4.3.2 Plot options for the matchit summary object

You can also send a matchit summary object to the `plot()` command, to obtain a summary of the balance on each covariate before and after matching. The summary() object must have been created using the option standardize=TRUE. The idea for this plot came from the "twang" package by McCaffrey, Ridgeway, and Morral.

4.3.2.1 Syntax

```
> s.out <- summary(object, standardize=TRUE, ...)

> plot(s.out, ...)
```

4.3.2.2 Arguments

- `interactive`: If `TRUE` (default), users can identify individual variables by clicking on the graph with the left mouse button.

4.3.2.3 Output Values

- Line plot of standardized differences in means before and after matching. Numbers plotted are those output by the summary() command in the sum.all and sum.matched objects.

4.4 `match.data()`: Extracting the Matched Data Set

4.4.1 Usage

To extract the matched data set for subsequent analyses from the output object (see Section 3.3), we provide the function `match.data()`. This is used as follows:

```
> m.data <- match.data(object, group = "all", distance = "distance",
                       weights = "weights", subclass = "subclass")
```

The output of the function `match.data()` is the original data frame where additional information about matching (i.e., distance measure as well as resulting weights and subclasses) is added, restricted to units that were matched.

4.4.2 Arguments

`match.data()` takes the following inputs:

1. `object` is the output object from `matchit()`. This is a required input.

2. `group` specifies for which matched group the user wants to extract the data. Available options are `"all"` (all matched units), `"treat"` (matched units in the treatment group), and `"control"` (matched units in the control group). The default is `"all"`.

3. `distance` specifies the variable name used to store the distance measure. The default is `"distance"`.

4. `weights` specifies the variable name used to store the resulting weights from matching. The default is `"weights"`. See Section 5.3 for more details on the weights.

5. `subclass` specifies the variable name used to store the subclass indicator. The default is `"subclass"`.

4.4.3 Examples

Here, we present examples for using `match.data()`. Users can run these commands by typing `demo(match.data)` at the R prompt. First, we load the Lalonde data,

```
> data(lalonde)
```

The next line performs nearest neighbor matching based on the estimated propensity score from the logistic regression,

```
> m.out1 <- matchit(treat ~ re74 + re75 + age + educ, data = lalonde,
+     method = "nearest", distance = "logit")
```

To obtain matched data, type the following command,

```
> m.data1 <- match.data(m.out1)
```

It is easy to summarize the resulting matched data,

```
> summary(m.data1)
```

To obtain matched data for the treatment or control group, specify the option `group` as follows,

```
> m.data2 <- match.data(m.out1, group = "treat")
> summary(m.data2)
> m.data3 <- match.data(m.out1, group = "control")
> summary(m.data3)
```

We can also use the function to return unmatched data:

```
> unmatched.data <- lalonde[!row.names(lalonde)%in%row.names(match.data(m.out1)),]
```

We can also specify different names for the subclass indicator, the weight variable, and the estimated distance measure. The following example first does a subclassification method, obtains the matched data with specified names for those three variables, and then print out the names of all variables in the resulting matched data.

```
> m.out2 <- matchit(treat ~ re74 + re75 + age + educ, data = lalonde,
+     method = "subclass")
> m.data4 <- match.data(m.out2, subclass = "block", weights = "w",
+     distance = "pscore")
> names(m.data4)
```

Chapter 5

Frequently Asked Questions

5.1 How do I Cite this Work?

If you use MatchIt, please cite

> Daniel Ho; Kosuke Imai; Gary King; and Elizabeth Stuart (2007), "Matching as Nonparametric Preprocessing for Reducing Model Dependence in Parametric Causal Inference," *Political Analysis* 15(3): 199-236, `http://gking.harvard.edu/files/abs/matchp-abs.shtml`.
>
> and
>
> Daniel Ho; Kosuke Imai; Gary King; and Elizabeth Stuart (2007b) "Matchit: Nonparametric Preprocessing for Parametric Causal Inference," *Journal of Statistical Software*, `http://gking.harvard.edu/matchit/`.

In addition, the `convex.hull` discard option is implemented via the `WhatIf` package (King and Zeng 2006, 2007; Stoll et al. 2005). Generalized linear distance measures are implemented via the `stats` package (Venables and Ripley 2002). Generalized additive distance measures are implemented via the `mcgv` package (Hastie and Tibshirani 1990). The neural network distance measure is implemented via the `nnet` package (Ripley 1996). The classification trees distance measure is implemented via the `rpart` package (Breiman et al. 1984). Full and optimal matching are implemented via the `optmatch` package (Hansen 2004). Genetic matching is implemented via the `Matching` package (Diamond and Sekhon 2005). Coarsened exact matching is implemented via the `cem` package (**??**).

5.2 What if My datasets Are Big and Are Taking Up Too Much Memory?

`matchit()` does not save the data set in its output object, but it does save a matrix of the covariates. `match.data()` will create a matched data set. One can eliminate the original

data set to save memory in R by `rm(name)`, where `name` is the name of the data set, after calling `match.data()`.

5.3 How Exactly are the Weights Created?

Each type of matching method can be thought of as creating groups of units with at least one treated unit and at least one control unit in each. In exact matching, subclassification, or full matching, these groups are the subclasses formed, and the number of treated and control units will vary quite a bit across subclasses. In nearest neighbor or optimal matching, the groups are the pairs (or sets) of treated and control units matched. In 1:1 nearest neighbor matching there will be one treated unit and one control unit in each group. In 2:1 nearest neighbor matching there will be one treated unit and two control units in each group. Unmatched units receive a weight of 0. All matched treated units receive a weight of 1. These weights are constructed to estimate the average treatment effect on the treated, with the control group essentially weighted to look like the treated group.

The weights for matched control units are formed as follows:

1. Within each group, each control unit is given a preliminary weight of n_{ti}/n_{ci}, where n_{ti} and n_{ci} are the number of treated and control units in group i, respectively.

2. If matching is done with replacement, each control unit's weight is added up across the groups in which it was matched.

3. The control group weights are scaled to sum to the number of uniquely matched control units.

With subclassification, when the analysis is done separately within each subclass and then aggregated up across the subclasses, these weights will generally not be used, but they may be used for full matching or nearest neighbor matching if the number of control units matched to each treated unit varies.

5.4 How Do I Create Observation Names?

Since the diagnostics often make use of the observation names of the data frame, you may find it helpful to specify observation names for the data input. Use the `row.names` command to achieve this. For example, to assign the names "Dan", "Kosuke", "Liz" and "Gary" to a data frame with the first four observations in the Lalonde data, type:

```
> test <- lalonde[1:4, ]
> row.names(test) <- c("Dan", "Kosuke", "Liz", "Gary")
> print(test)
       age educ black hisp married nodegr re74 re75  re78 u74 u75 treat
Dan     37   11     1    0       1      1    0    0  9930   1   1     1
```

```
Kosuke  22    9     0    1       0      1    0    0  3596   1   1     1
Liz     30   12     1    0       0      0    0    0 24910   1   1     1
Gary    27   11     1    0       0      1    0    0  7506   1   1     1
```

5.5 How Can I See Outcomes of Matched Pairs?

To obtain outcomes of matched pairs, recall that the original dataset has unique row names corresponding to each of the observations. The row names of `match.matrix` correspond to the names of the treated, and each of the cells corresponds to a name of matched controls. So to obtain matched outcomes, you can use:

```
cbind(lalonde[row.names(foo$match.matrix),"re78"], lalonde[foo$match.matrix,"re78"])
```

5.6 How Do I Ensure Replicability As MatchIt Versions Develop?

As the literature on matching techniques is rapidly evolving, MatchIt will strive to incorporate new developments. MatchIt is thereby an evolving program. Users may be concerned that analysis written in a particular version may not be compatible with newer versions of the program. The primary way to ensure that replication archives remain valid is to record the version of MatchIt that was used in the analysis. Our website maintains binaries of all public release versions, so that researchers can replicate results exactly with the appropriate version (for Unix-based platforms, see http://gking.harvard.edu/src/contrib/; for windows, see http://gking.harvard.edu/bin/windows/contrib/).

In addition, users may find it helpful to install packages with version control, using the `installWithVers` command with `install.packages`. So for example, in the windows R console, users may download the appropriate version from our website and install the package with version control by:

```
install.packages(choose.files('',filters=Filters[c('zip','All'),]),
                 .libPaths()[1],installWithVers=T,CRAN=NULL)
```

`R CMD INSTALL` similarly permits users to specify this version using the `--with-package-versions` option. After having specified version control, different versions of the program may be called as necessary. Similar advice may also be appropriate for version control for R more generally.

5.7 How Do I Use My Own Distance Measure with MatchIt ?

A vector of your own distance measure can be used by specifying it as the input for `distance` option in `matchit()`.

5.8 What Do I Do about Missing Data?

MatchIt requires complete data sets, with no missing values (other than potential outcomes of course). If there are missing values in the data set, imputation techniques should be used first to fill in ("impute") the missing values (both covariates and outcomes), or the analysis should be done using only complete cases (which we do not in general recommend). For imputation software, see Amelia at (http://gking.harvard.edu/stats.shtml) or other programs at http://www.multiple-imputation.com. For more information on missing data and imputation methods, see King et al. (2001).

5.9 Why Preprocessing?

The purpose of matching is to approximate an experimental template, where the matching procedure approximates blocking prior to random treatment assignment in order to balance covariates between treatment and control groups. Separation of the estimation procedure into two steps simulates the research design of an experiment, where no information on outcomes is known at the point of experimental design and randomization. The separation of the balancing process in MatchIt from the analysis process afterward helps keep clear the goal of balancing control and treatment groups and makes it less likely that the user will inadvertently cook the books in his or her favor.

Chapter 6

What's New?

- **2.4-18** (April 26, 2011): JSS version, no change in code.
- **2.4-17** (April 2, 2011): a minor documentation fix.
- **2.4-16** (January 8, 2011): a bug fix for user defined distance.
- **2.4-15** (December 11, 2010): a bug fix in the mahalanobis matching.
- **2.4-14** (August 12, 2010): a bug fix in `match.data()` so that it can be called within a function (thanks to Ajay Shah, George Baah, and Ben Dominique); MatchItnow does not specify digits for printing (thanks to Chris Hane); A summary table of matched data is now stored in the output (thanks to George Baah)
- **2.4-11** (June 25, 2009): More flexible inputs in plotting.
- **2.4-10** (February 2, 2009): Minor documentation fixes
- **2.4-8,2.4-9** (January 29, 2009): Minor documentation fixes
- **2.4-7** (August 4, 2008): Fixed minor bug in subclassification (thanks to Ben Domingue)
- **2.4-6** (July 21, 2008): Improved summary object for exact matching (thanks to Andrew Stokes)
- **2.4-5** (July 20, 2008): Fixed a minor bug.
- **2.4-4** (July 18, 2008): Fixed another bug with regard to the discard option (thanks to Ben Dominique).
- **2.4-3** (July 18, 2008): Fixed a bug in full matching regarding the discard option (thanks to Ben Dominique). Some updates of documentation regarding coarsened exact matching (2.4-1 and 2.4-2).
- **2.4** (June 12, 2008): Included coarsened exact matching; documentation bug fixes (thanks to Will Lowe)

- **2.3-1** (October 11, 2007): Stable release for R 2.6. Documentation improved. Some minor bug fixes and improvements.
- **2.2-14** (September 2, 2007): Stable release for R 2.5. Documentation improved for full matching. (Thanks to Langche Zeng)
- **2.2-13** (April 10, 2007): Stable release for R 2.4. Additional fix to package dependencies. Bug fix for summary().
- **2.2-12** (April 6, 2007): Stable release for R 2.4. Fix to package dependencies.
- **2.2-11** (July 13, 2006): Stable release for R 2.3. Fix to ensure summary() command works with character variables in dataframe (thanks to Daniel Gerlanc).
- **2.2-10** (May 9, 2006): Stable release for R 2.3. A bug fix in `demo(analysis)` (thanks to Julia Gray).
- **2.2-9** (May 3, 2006): Stable release for R 2.3. A minor change to DESCRIPTION file.
- **2.2-8** (May 1, 2006): Stable release for R 2.3. Removed dependency on Zelig (thanks to Dave Kane).
- **2.2-7** (April 11, 2006): Stable release for R 2.2. Error message for missing values in the data frame added (thanks to Olivia Lau).
- **2.2-6** (April 4, 2006): Stable release for R 2.2. Bug fixes related to `reestimate` in `matchit()` and `match.data()` (thanks to Ani Ruhil and Claire Aussems).
- **2.2-5** (December 7, 2005): Stable release for R 2.2. Changed URL of `WhatIf` to CRAN.
- **2.2-4** (December 3, 2005): Stable release for R 2.2. User's own distance measure can be used with MatchIt (thanks to Nelson Lim).
- **2.2-3** (November 18, 2005): Stable release for R 2.2. standardize option added to full matching and subclass (thanks to Jeronimo Cortina).
- **2.2-2** (November 9, 2005): Stable release for R 2.2. `optmatch` package now on CRAN. Changed URL for that package.
- **2.2-1** (November 1, 2005): Stable release for R 2.2. balance measures based on empirical CDF are added as a new option `standardize` in `summary()`.
- **2.1-4** (October 14, 2005): Stable release for R 2.2. strictly empirical (no interpolation) quantile-quantile functions and plots are used.
- **2.1-3** (September 27, 2005): Stable release for R 2.1. automated the installation of optional packages. fixed a coding error in `summary()`, the documentation edited.

- **2.1-2** (September 27, 2005): Stable release for R 2.1. minor changes to file names, the option `"whichxs"` added to the `plot()`, major editing of the documentation.
- **2.1-1** (September 16, 2005): Stable release for R 2.1. Genetic matching added.
- **2.0-1** (August 29, 2005): Stable release for R 2.1. Major revisions including some syntax changes. Statistical tests are no longer used for balance checking, which are now based on the empirical covariate distributions (e.g., quantile-quantile plot).
- **1.0-2** (August 10, 2005): Stable release for R 2.1. Minor bug fixes (Thanks to Bart Bonikowski).
- **1.0-1** (January 3, 2005): Stable release for R 2.0. The first official version of MatchIt

Bibliography

Abadie, A. and Imbens, G. W. (2007), "Bias-Corrected Matching Estimators for Average Treatment Effects," http://ksghome.harvard.edu/ aabadie/research.html.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984), *Classification and Regression Trees*, New York, New York: Chapman & Hall.

Dehejia, R. H. and Wahba, S. (1999), "Causal Effects in Nonexperimental Studies: Re-Evaluating the Evaluation of Training Programs," *Journal of the American Statistical Association*, 94, 1053–62.

Diamond, A. and Sekhon, J. (2005), "Genetic Matching for Estimating Causal Effects: A New Method of Achieving Balance in Observational Studies," http://jsekhon.fas.harvard.edu/.

Gu, X. and Rosenbaum, P. R. (1993), "Comparison of multivariate matching methods: structures, distances, and algorithms," *Journal of Computational and Graphical Statistics*, 2, 405–420.

Hansen, B. B. (2004), "Full Matching in an Observational Study of Coaching for the SAT," *Journal of the American Statistical Association*, 99, 609–618.

Hastie, T. J. and Tibshirani, R. (1990), *Generalized Additive Models*, London: Chapman Hall.

Ho, D., Imai, K., King, G., and Stuart, E. (2007), "Matching as Nonparametric Preprocessing for Reducing Model Dependence in Parametric Causal Inference," *Political Analysis*, 15, 199–236, http://gking.harvard.edu/files/abs/matchp-abs.shtml.

Ho, D. E., Imai, K., King, G., and Stuart, E. A. (Forthcoming), "MatchIt: Nonparametric Preprocessing for Parametric Causal Inference," *Journal of Statistical Software*, http://gking.harvard.edu/matchit.

Imai, K. (2005), "Do Get-Out-The-Vote Calls Reduce Turnout? The Importance of Statistical Methods for Field Experiments," *American Political Science Review*, 99, 283–300.

Imai, K., King, G., and Lau, O. (2006), "Zelig: Everyone's Statistical Software," http://gking.harvard.edu/zelig.

Imai, K., King, G., and Stuart, E. (2008), "Misunderstandings Among Experimentalists and Observationalists about Causal Inference," *Journal of the Royal Statistical Society, Series A*, 171, part 2, 481–502, http://gking.harvard.edu/files/abs/matchse-abs.shtml.

Imai, K. and van Dyk, D. A. (2004), "Causal Inference with General Treatment Treatment Regimes: Generalizing the Propensity Score," *Journal of the American Statistical Association*, 99, 854–866.

King, G., Honaker, J., Joseph, A., and Scheve, K. (2001), "Analyzing Incomplete Political Science Data: An Alternative Algorithm for Multiple Imputation," *American Political Science Review*, 95, 49–69, http://gking.harvard.edu/files/abs/evil-abs.shtml.

King, G. and Zeng, L. (2006), "The Dangers of Extreme Counterfactuals," *Political Analysis*, 14, 131–159, http://gking.harvard.edu/files/abs/counterft-abs.shtml.

— (2007), "When Can History Be Our Guide? The Pitfalls of Counterfactual Inference," *International Studies Quarterly*, 183–210, http://gking.harvard.edu/files/abs/counterf-abs.shtml.

Lalonde, R. (1986), "Evaluating the Econometric Evaluations of Training Programs," *American Economic Review*, 76, 604–620.

Ripley, B. (1996), *Pattern Recognition and Neural Networks*, Cambridge Univeristy Press.

Rosenbaum, P. R. (2002), *Observational Studies, 2nd Edition*, New York, NY: Springer Verlag.

Stoll, H., King, G., and Zeng, L. (2005), "WhatIf: Software for Evaluating Counterfactuals," *Journal of Statistical Software*, 15, http://www.jstatsoft.org/index.php?vol=15.

Venables, W. N. and Ripley, B. D. (2002), *Modern Applied Statistics with S*, Springer-Verlag, 4th ed.