Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. http://www.jstatsoft.org/

Multivariate and Propensity Score Matching Software with Automated Balance Optimization: The Matching package for R

Jasjeet S. Sekhon
UC Berkeley

Abstract

`Matching` is an R package which provides functions for multivariate and propensity score matching and for finding optimal covariate balance based on a genetic search algorithm. A variety of univariate and multivariate metrics to determine if balance actually has been obtained are provided. The underlying matching algorithm is written in C++, makes extensive use of system BLAS and scales efficiently with dataset size. The genetic algorithm which finds optimal balance is parallelized and can make use of multiple CPUs or a cluster of computers. A large number of options are provided which control exactly how the matching is conducted and how balance is evaluated.

Keywords: propensity score matching, multivariate matching, genetic optimization, causal inference, R.

1. Introduction

The R package **Matching** implements a variety of algorithms for multivariate matching including propensity score, Mahalanobis, inverse variance and Genetic Matching (GenMatch). The last of these, Genetic Matching, is a method which automatically finds the set of matches which minimize the discrepancy between the distribution of potential confounders in the treated and control groups—i.e., covariate balance is maximized. The package enables a wide variety of matching options including matching with or without replacement, bias adjustment, different methods for handling ties, exact and caliper matching, and a method for the user to fine tune the matches via a general restriction matrix. Variance estimators include the usual Neyman standard errors (which condition on the matched data), Abadie and Imbens (2006) standard errors which account for the (asymptotic) variance induced by the matching procedure itself, and robust variances which do not assume a homogeneous causal effect.

Electronic copy available at: http://ssrn.com/abstract=1009044

The package provides a set of functions to do the Matching (`Match`) and to evaluate how good covariate balance is before and after matching (`MatchBalance`). The `GenMatch` function finds optimal balance using multivariate matching where a genetic search algorithm determines the weight each covariate is given. Balance is determined by examining cumulative probability distribution functions of a variety of standardized statistics. By default, these statistics include paired t-tests, univariate and multivariate Kolmogorov-Smirnov (KS) tests. A variety of descriptive statistics based on empirical-QQ plots are also offered. The statistics are not used to conduct formal hypothesis tests, because no measure of balance is a monotonic function of bias in the estimand of interest and because we wish to maximize balance without limit. `GenMatch` can maximize balance based on a variety of pre-defined loss functions or any loss function the user may wish to provide.

In the next section I briefly offer some background material on both the Rubin Causal Model and matching methods. Section 3 provides an overview of the `Matching` package with examples. Section 4 concludes.

2. Background on Matching

Matching has become an increasingly popular method of causal inference in many fields including statistics (Rubin 2006; Rosenbaum 2002), medicine (Christakis and Iwashyna 2003; Rubin 1997), economics (Abadie and Imbens 2006; Galiani, Gertler, and Schargrodsky 2005; Dehejia and Wahba 2002, 1999), political science (Bowers and Hansen 2005; Herron and Wand forthcoming; Imai 2005; Sekhon 2004), sociology (Morgan and Harding 2006; Diprete and Engelhardt 2004; Winship and Morgan 1999; Smith 1997) and even law (Rubin 2001). There is, however, no consensus on how exactly matching ought to be done and how to measure the success of the matching procedure. A wide variety of matching procedures have been proposed, and `Matching` implements many of them.

When using matching methods to estimate causal effects, a central problem is deciding how best to perform the matching. Two common approaches are propensity score matching (Rosenbaum and Rubin 1983) and multivariate matching based on Mahalanobis distance (Cochran and Rubin 1973; Rubin 1979, 1980). Matching methods based on the propensity score (estimated by logistic regression), Mahalanobis distance or a combination of the two have appealing theoretical properties if covariates have ellipsoidal distributions—e.g., distributions such as the normal or t. If the covariates are so distributed, these methods (more generally affinely invariant matching methods[1]) have the property of "equal percent bias reduction" (EPBR) (Rubin 1976a,b; Rubin and Thomas 1992).[2] This property is formally defined in Appendix A. When this property holds, matching will reduce bias in all linear combinations of the covariates. If the EPBR property does not hold, then, in general, matching will increase the bias of some linear functions of the covariates even if all univariate means are closer in the matched data than the unmatched (Rubin 1976a). Unfortunately, the EPBR property

[1] Affine invariance means that the matching output is invariant to matching on X or an affine transformation of X.

[2] The EPBR results of Rubin and Thomas (1992) have been extended by Rubin and Stuart (2006) to the case of discriminant mixtures of proportional ellipsoidally symmetric (DMPES) distributions. This extension is important, but it is restricted to a limited set of mixtures.

Electronic copy available at: http://ssrn.com/abstract=1009044

rarely holds with actual data.

A significant shortcoming of common matching methods such as Mahalanobis distance and propensity score matching is that they may (and in practice, frequently do) make balance worse across measured potential confounders. These methods may make balance worse, in practice, even if covariates are distributed ellipsoidally because in a given finite sample there may be departures from an ellipsoidal distribution. Moreover, if covariates are neither ellipsoidally symmetric nor are mixtures of discriminant mixtures of proportional ellipsoidally symmetric (DMPES) distributions, propensity score matching has good theoretical properties if and only if the true propensity score model is known and the sample size is large.

These limitations often surprise applied researchers. Because of the limited theoretical properties for matching when the propensity score is not known, one approach is to algorithmically impose additional properties, and this is the approach used by Genetic Matching.

Sekhon (2006a) and Diamond and Sekhon (2005) propose a matching algorithm, Genetic Matching (GenMatch), which maximizes the balance of observed covariates between treated and control groups. GenMatch is a generalization of propensity score and Mahalanobis distance matching, and it has been used by a variety of researchers (e.g., Bonney, Canes-Wrone, and Minozzi 2007; Brady and Hui 2006; Gilligan and Sergenti 2006; Gordon and Huber 2007; Herron and Wand forthcoming; Morgan and Harding 2006; Lenz and Ladd 2006; Park 2006; Raessler and Rubin 2005). The algorithm uses a genetic algorithm (Mebane and Sekhon 1998; Sekhon and Mebane 1998) to optimize balance as much as possible given the data. The method is nonparametric and does not depend on knowing or estimating the propensity score, but the method is improved when a propensity score is incorporated. Diamond and Sekhon (2005) use this algorithm to show that the long running debate between Dehejia and Wahba (2002; 1997; 1999; Dehejia 2005) and Smith and Todd (2005b,a, 2001) is largely a result of researchers using models which do not produce good balance—even if some of the models get close by chance to the experimental benchmark of interest. They show that Genetic Matching is able to quickly find good balance and to reliably recover the experimental benchmark.

The core motivation for all matching methods is the Rubin Causal Model which I discuss next followed by details on Mahalanobis, propensity score and Genetic Matching.

2.1. Rubin Causal Model

The Rubin causal model conceptualizes causal inference in terms of potential outcomes under treatment and control, only one of which is observed for each unit (Holland 1986; Splawa-Neyman 1923; Rubin 1974, 1978, 1990). A causal effect is defined as the difference between an observed outcome and its counterfactual.

Let Y_{i1} denote the potential outcome for unit i if the unit receives treatment, and let Y_{i0} denote the potential outcome for unit i in the control regime. The treatment effect for observation i is defined by $\tau_i = Y_{i1} - Y_{i0}$. Causal inference is a missing data problem because Y_{i1} and Y_{i0} are never both observed. Let T_i be a treatment indicator equal to 1 when i is in the treatment regime and 0 otherwise. The observed outcome for observation i is then $Y_i = T_i Y_{i1} + (1 - T_i) Y_{i0}$.

In principle, if assignment to treatment is randomized, causal inference is straightforward because the two groups are drawn from the same population by construction, and treatment

assignment is independent of all baseline variables. As the sample size grows, observed and unobserved baseline variables are balanced across treatment and control groups with arbitrarily high probability, because treatment assignment is independent of Y_0 and Y_1—i.e., following Dawid's (1979) notation, $\{Y_{i0}, Y_{i1} \perp\!\!\!\perp T_i\}$. Hence, for $j = 0, 1$

$$E(Y_{ij} \mid T_i = 1) = E(Y_{ij} \mid T_i = 0) = E(Y_i \mid T_i = j)$$

Therefore, the average treatment effect (ATE) can be estimated by:

$$\begin{aligned} \tau &= E(Y_{i1} \mid T_i = 1) - E(Y_{i0} \mid T_i = 0) \\ &= E(Y_i \mid T_i = 1) - E(Y_i \mid T_i = 0) \end{aligned} \tag{1}$$

Equation 1 is estimable in an experimental setting because observations in treatment and control groups are exchangeable.[3] In the simplest experimental setup, individuals in both groups are equally likely to receive the treatment, and hence assignment to treatment will not be associated with the outcome. Even in an experimental setup, much can go wrong which requires statistical correction (e.g., Barnard, Frangakis, Hill, and Rubin 2003).

In an observational setting, covariates are almost never balanced across treatment and control groups because the two groups are not ordinarily drawn from the same population. Thus, a common quantity of interest is the average treatment effect for the treated (ATT):

$$\tau \mid (T = 1) \quad = \quad E(Y_{i1} \mid T_i = 1) - E(Y_{i0} \mid T_i = 1). \tag{2}$$

Equation 2 cannot be directly estimated because Y_{i0} is not observed for the treated. Progress can be made by assuming that selection into treatment depends on observable covariates X. Following Rosenbaum and Rubin (1983), one can assume that conditional on X, treatment assignment is unconfounded ($\{Y_0, Y_1 \perp\!\!\!\perp T\} \mid X$) and that there is overlap: $0 < Pr(T = 1 \mid X) < 1$. Together, unconfoundedness and overlap constitute a property known as strong ignorability of treatment assignment which is necessary for identifying the average treatment effect. Heckman, Ichimura, Smith, and Todd (1998) show that for ATT, the unconfoundedness assumption can be weakened to mean independence: $E\left(Y_{ij} \mid T_i, X_i\right) = E\left(Y_{ij} \mid X_i\right)$.[4] The overlap assumption for ATT only requires that the support of X for the treated be a subset of the support of X for control observations.

Then, following Rubin (1974, 1977) we obtain

$$E(Y_{ij} \mid X_i, T_i = 1) = E(Y_{ij} \mid X_i, T_i = 0) = E(Y_i \mid X_i, T_i = j). \tag{3}$$

By conditioning on observed covariates, X_i, treatment and control groups are exchangeable. The average treatment effect for the treated is estimated as

$$\tau \mid (T = 1) \quad = \quad E\left\{E(Y_i \mid X_i, T_i = 1) - E(Y_i \mid X_i, T_i = 0) \mid T_i = 1\right\}, \tag{4}$$

where the outer expectation is taken over the distribution of $X_i \mid (T_i = 1)$ which is the distribution of baseline variables in the treated group.

[3] It is standard practice to assume the Stable Unit Treatment Value assumption, also known as SUTVA (Holland 1986; Rubin 1978). SUTVA requires that the treatment status of any unit be independent of potential outcomes for all other units, and that treatment is defined identically for all units.

[4] Also see Abadie and Imbens (2006).

The most straightforward and nonparametric way to condition on X is to exactly match on the covariates. This is an old approach going back to at least Fechner (1966 [1860]), the father of psychophysics. This approach fails in finite samples if the dimensionality of X is large or if X contains continuous covariates. Thus, in general, alternative methods must be used.

2.2. Mahalanobis and Propensity Score Matching

The most common method of multivariate matching is based on Mahalanobis distance (Cochran and Rubin 1973; Rubin 1979, 1980). The Mahalanobis distance between any two column vectors is:

$$md(X_i, X_j) \quad = \left\{(X_i - X_j)' S^{-1} (X_i - X_j)\right\}^{\frac{1}{2}}$$

where S is the sample covariance matrix of X. To estimate ATT by matching with replacement, one matches each treated unit with the M closest control units, as defined by this distance measure, $md()$.[5] If X consists of more than one continuous variable, multivariate matching estimates contain a bias term which does not asymptotically go to zero at rate $\sqrt{n}$ (Abadie and Imbens 2006).

An alternative way to condition on X is to match on the probability of assignment to treatment, known as the propensity score.[6] As one's sample size grows large, matching on the propensity score produces balance on the vector of covariates X (Rosenbaum and Rubin 1983).

Let $e(X_i) \equiv Pr(T_i = 1 \mid X_i) = E(T_i \mid X_i)$, defining $e(X_i)$ to be the propensity score. Given $0 < Pr(T_i \mid X_i) < 1$ and $Pr(T_1, T_2, \cdots T_N \mid X_1, X_2, \cdots X_N) = \Pi_{i=1}^{N} e(X_i)^{T_i} (1 - e(X_i))^{(1-T_i)}$, Rosenbaum and Rubin (1983) prove that

$$\tau \mid (T = 1) \quad = \quad E\left\{E(Y_i \mid e(X_i), T_i = 1) - E(Y_i \mid e(X_i), T_i = 0) \mid T_i = 1\right\},$$

where the outer expectation is taken over the distribution of $e(X_i) \mid (T_i = 1)$. Since the propensity score is generally unknown, it must be estimated.

Propensity score matching involves matching each treated unit to the nearest control unit on the unidimensional metric of the propensity score vector. If the propensity score is estimated by logistic regression, as is typically the case, much is to be gained by matching not on the predicted probabilities (bounded between zero and one) but on the linear predictor $\hat{\mu} \equiv X\hat{\beta}$. Matching on the linear predictor avoids compression of propensity scores near zero and one. Moreover, the linear predictor is often more nearly normally distributed which is of some importance given the EPBR results if the propensity score is matched on along with other covariates.

Mahalanobis distance and propensity score matching can be combined in various ways (Rubin 2001; Rosenbaum and Rubin 1985). It is useful to combine the propensity score with Mahalanobis distance matching because propensity score matching is particularly good at minimizing the discrepancy along the propensity score and Mahalanobis distance is particularly good at minimizing the distance between individual coordinates of X (orthogonal to the propensity score) (Rosenbaum and Rubin 1985).

[5] Alternatively one can do optimal full matching (Hansen 2004; Hansen and Klopfer 2006; Rosenbaum 1989, 1991) instead of the 1-to-N matching with replacement which I focus on in this article. This decision is a separate one from the choice of a distance metric.

[6] The first estimator of treatment effects to be based on a weighted function of the probability of treatment was the Horvitz-Thompson statistic (Horvitz and Thompson 1952).

2.3. Genetic Matching

The idea underlying the GenMatch algorithm is that if Mahalanobis distance is not optimal for achieving balance in a given dataset, one should be able to search over the space of distance metrics and find something better. One way of generalizing the Mahalanobis metric is to include an additional weight matrix:

$$d(X_i, X_j) \quad = \left\{ (X_i - X_j)' \left(S^{-1/2}\right)' W S^{-1/2} (X_i - X_j) \right\}^{\frac{1}{2}}$$

where W is a $k \times k$ positive definite weight matrix and $S^{1/2}$ is the Cholesky decomposition of S which is the variance-covariance matrix of X.[7]

Note that if one has a good propensity score model, one should include it as one of the covariates in `GenMatch`. If this is done, both propensity score matching and Mahalanobis matching can be considered special limiting cases of `GenMatch`. If the propensity score contains all of the relevant information in a given sample, the other variables will be given zero weight.[8] And `GenMatch` will converge to Mahalanobis distance if that proves to be the appropriate distance measure.

`GenMatch` is an affinely invariant matching algorithm that uses the distance measure $d()$, in which all elements of W are zero except down the main diagonal. The main diagonal consists of k parameters which must be chosen. Note that if each of these k parameters are set equal to 1, $d()$ is the same as Mahalanobis distance.

The choice of setting the non-diagonal elements of W to zero is made for reasons of computational power alone. The optimization problem grows exponentially with the number of free parameters. It is important that the problem be parameterized so as to limit the number of parameters which must be estimated.

This leaves the problem of how to choose the free elements of W. Many loss criteria recommend themselves, and `GenMatch` provides a number the user can choose from via the `fit.func` and `loss` options of `GenMatch`. By default, cumulative probability distribution functions of a variety of standardized statistics are used as balance metrics and are optimized without limit. The default standardized statistics are paired t-tests and nonparametric KS tests.

The statistics are not used to conduct formal hypothesis tests, because no measure of balance is a monotonic function of bias in the estimand of interest and because we wish to maximize balance without limit. Descriptive measures of discrepancy generally ignore key information related to bias which is captured by probability distribution functions of standardized test statistics. For example, using several descriptive metrics, one is unable to recover reliably the experimental benchmark in a testbed dataset for matching estimators (Dehejia and Wahba 1999). And these metrics, unlike those based on optimized distribution functions, perform poorly in a series of Monte Carlo sampling experiments just as one would expect given their properties. For details see Sekhon (2006a).

By default, `GenMatch` attempts to minimize a measure of the maximum observed discrepancy between the matched treated and control covariates at every iteration of optimization. For a given set of matches resulting from a given W, the loss is defined as the minimum p-value

[7] The Cholesky decomposition is parameterized such that $S = LL'$, $S^{1/2} = L$. In other words, L is a lower triangular matrix with positive diagonal elements.

[8] Technically, the other variables will be given weights just large enough to ensure that the weight matrix is positive definite.

observed across a series of standardized statistics. The user may specify exactly what tests are done via the `BalanceMatrix` option. Examples are offered in Section 3.

Conceptually, the algorithm attempts to minimize the largest observed covariate discrepancy at every step. This is accomplished by maximizing the smallest p-value at each step.[9] Because `GenMatch` is minimizing the maximum discrepancy observed at each step, it is minimizing the infinity norm. This property holds even when, because of the distribution of X, the EPBR property does not hold. Therefore, if an analyst is concerned that matching may increase the bias in some linear combination of X even if the means are reduced, `GenMatch` allows the analyst to put in the loss function all of the linear combinations of X which may be of concern. Indeed, any nonlinear function of X can also be included in the loss function, which would ensure that bias in some nonlinear functions of X is not made inordinately large by matching.

The default `GenMatch` loss function does allow for imbalance in functions of X to worsen as long as the maximum discrepancy is reduced. Hence, it is important that the maximum discrepancy be small—i.e., that the smallest p-value be large. p-values conventionally understood to signal balance (e.g., 0.10), may be too low to produce reliable estimates. After GenMatch optimization, the p-values from these balance tests cannot be interpreted as true probabilities because of standard pre-test problems, but they remain useful measures of balance. Also, we are interested in maximizing the balance in the current sample so a hypothesis test for balance is inappropriate.

The optimization problem described above is difficult and irregular, and the genetic algorithm implemented in the `rgenoud` package (Mebane and Sekhon 1998) is used to conduct the optimization. Details of the algorithm are provided in Sekhon and Mebane (1998).

`GenMatch` is shown to have better properties than the usual alternative matching methods both when the EPBR property holds and when it does not (Sekhon 2006a; Diamond and Sekhon 2005). Even when the EPBR property holds and the mapping from X to Y is linear, GenMatch has better efficiency—i.e., lower mean square error (MSE)—in finite samples. When the EPBR property does not hold as it generally does not, GenMatch retains appealing properties and the differences in performance between GenMatch and the other matching methods can become substantial both in terms of bias and MSE reduction. In short, at the expense of computer time, GenMatch dominates the other matching methods in terms of MSE when assumptions required for EPBR hold and, even more so, when they do not.

GenMatch is able to retain good properties even when EPBR does not hold because a set of constraints is imposed by the loss function optimized by the genetic algorithm. The loss function depends on a large number of functions of covariate imbalance across matched treatment and control groups. Given these measures, `GenMatch` will optimize covariate balance.

3. Package Overview and Examples

The three main functions in the package are `Match`, `MatchBalance` and `GenMatch`. The first function, `Match`, performs multivariate and propensity score matching. It is intended to be

[9] More precisely lexical optimization will be done: all of the balance statistics will be sorted from the most discrepant to the least and weights will be picked which minimize the maximum discrepancy. If multiple sets of weights result in the same maximum discrepancy, then the second largest discrepancy is examined to choose the best weights. The processes continues iteratively until ties are broken.

used in conjunction with the `MatchBalance` function which checks if the results of `Match` have actually achieved balance on a set of covariates. `MatchBalance` can also be used before any matching to determine how balanced the raw data is. If one wants to do propensity score matching, one should estimate the propensity model before calling `Match`, and then send `Match` the propensity score to use. The `GenMatch` function can be used to *automatically find balance* by the use of a genetic search algorithm which determines the optimal weight to give each covariate.

Next, I present a set of propensity score (pscore) models which perform better as adjustments are made to them after the output of `MatchBalance` is examined. I then provide an example using `GenMatch`.

3.1. Propensity Score Matching Example

In order to do propensity score matching, the work flow is to first estimate a propensity score using, for example, `glm` if one wants to estimate a propensity score using logistic regression. A number of alternative methods of estimating the propensity score, such as General Additive Models (GAMs), are possible. After the propensity score has been estimated, one calls `Match` to perform the matching and `MatchBalance` to examine how well the matching procedure did in producing balance. If the balance results printed by `MatchBalance` are not good enough, one would go back and change either the propensity score model or some parameter of how the matching is done—e.g., change from 1-to-3 matching to 1-to-1 matching.

The following example is adopted from the documentation of the `Match` function. The example uses the LaLonde (1986) experimental data which is based on a nationwide job training experiment. The observations are individuals, and the outcome of interest is real earnings in 1978. There are eight baseline variables age (`age`), years of education (`educ`), real earnings in 1974 (`re74`), real earnings in 1975 (`re75`), and a series of indicator variables. The indicator variables are black (`black`), Hispanic (`hisp`), married (`married`) and lack of a high school diploma (`nodegr`).

```
library(Matching)
data(lalonde)

#save data objects
#
Y  <- lalonde$re78   #the outcome of interest
Tr <- lalonde$treat  #the treatment of interest

# First attempt at creating a propensity score model
#
glm1  <- glm(treat~age + educ + black +
             hisp + married + nodegr + re74  + re75,
             family=binomial, data=lalonde)

# one-to-one matching with replacement
# Estimating the treatment effect on the treated:
#   the "estimand" option defaults to ATT.
#
```

```
rr1  <- Match(Y=Y, Tr=Tr, X=glm1$fitted);
#
#we could do 'summary(rr1)' so see results, but wait until we have balance!

# Let's check for balance
MatchBalance(treat~age + I(age^2) + educ + I(educ^2) + black +
             hisp + married + nodegr + re74  + I(re74^2) + re75 + I(re75^2) +
             u74 + u75 + I(re74*re75) + I(age*nodegr) + I(educ*re74) + I(educ*re75),
             match.out=rr1, nboots=1000, data=lalonde)
```

The only command here which produces output is the call to `MatchBalance`. If we wanted to see the results from the call to `Match` which would display the causal estimate and its standard error we could do `summary(rr1)`, but it is best to wait until we have achieved satisfactory balance before looking at the estimates. To this end, `Match` does not even need to be provided with an outcome variable—i.e., `Y`—in order to work. Matches can be found and balance evaluated without knowledge of `Y`. Indeed, this is to be preferred so that the design stage of the observational study can be clearly separated from the estimation stage as is the case with experiments.

In the example above, the call to `glm` estimates a simple propensity score model and the syntax of this procedure is covered in the R documentation. Then a call to `Match` is made which relies heavily on the function's default behavior because only three options are explicitly provided: a vector (`Y`) containing the outcome variable, a vector (`Tr`) containing the treatment status of each observation—i.e., either a zero or one—and a matrix (`X`) containing the variables to be matched on which in this case is simply the propensity score. By default `Match` does 1-to-1 matching with replacement, and estimates ATT. The estimand is chosen via the `estimand` option, as in `estimand="ATE"` to estimate the average treatment effect. The ratio of treated to control observations is determined by the the `M` option and this ratio is by default set to 1. And whether matching should be done with replacement is controlled by the logical argument `replace` which defaults to `TRUE` for matching with replacement.

Ties are by default handled deterministically (Abadie and Imbens 2006) and this behavior is controlled by the `ties` option. By default `ties==TRUE`. If, for example, one treated observation matches more than one control observation, the matched dataset will include the multiple matched control observations and the matched data will be weighted to reflect the multiple matches. The sum of the weighted observations will still equal the original number of observations. If `ties==FALSE`, ties will be randomly broken. This in general is not a good idea because the variance of `Y` will be underestimated. But if the dataset is large and there are many ties between potential matches, setting `ties=FALSE` often results in a rather large speedup with negligible bias. Whether two potential matches are close enough to be considered tied, is controlled by the `distance.tolerance` option.

All of this default behavior of `Match` means that the simple command

```
m1 = Match(Y=Y, Tr=Tr, X=glm1$fitted)
```

is equivalent to

```
m1 = Match(Y=Y, Tr=Tr, X=glm1$fitted, estimand="ATT", M=1, ties=TRUE,
           replace=TRUE)
```

We generally want to test for balance for more functions of the data than we include in our propensity score model. In this case, note that our call to `MatchBalance` tests for many more functions of the confounders than we included in the propensity score model.

The formula used in the call to `MatchBalance` does *not* estimate any model.[10] The formula is simply an efficient way to use the R modeling language to list the variables we wish to obtain univariate balance statistics on. The dependent variable in the formula is the treatment indicator.

The propensity score model is different from the balance statistics which are requested from `MatchBalance`. In general, one does **not** need to include all of the functions one wants to test balance on in the propensity score model. Indeed, doing so sometimes results in *worse* balance. Generally, one should request balance statistics on more higher-order terms and interactions than were included in the propensity score used to conduct the matching itself.

Aside from the formula, three additional arguments were given to the `MatchBalance` call. The `match.out` option is used to provide the output object from the previous call to `Match`. If this object is provided, `MatchBalance` will provide balance statistics for both before and after matching, otherwise balance statistics will only be provided for the unmatched raw dataset. The `nboots` option determines the number of bootstrap samples to be run. If zero, no bootstraps are done. Bootstrapping is highly recommended because the bootstrapped Kolmogorov-Smirnov test, unlike the standard test, provides correct coverage even when there are point masses in the distributions being compared (Abadie 2002). At least 500 `nboots` (preferably 1000) are recommended for publication quality p-values. And finally, the `data` argument expects a data frame which contains all of the variables in the formula. If a data frame is not provided, the variables are obtained via lexical scoping.

For each term included into the modeling equation provided as the first argument to `MatchBalance`, detailed balance statistics are produced. Let's first consider the output for `nodegr`:

```
***** (V8) nodegr *****
                        Before Matching          After Matching
mean treatment........    0.70811                  0.70811
mean control..........    0.83462                  0.76757

mean std eQQ diff.....   0.063254                 0.021802
med  std eQQ diff.....   0.063254                 0.021802
max  std eQQ diff ....    0.12651                 0.043605

mean raw eQQ diff.....    0.12432                 0.043605
med  raw eQQ diff.....          0                        0
max  raw eQQ diff.....          1                        1

var ratio (Tr/Co).....     1.4998                   1.1585
T-test p-value........  0.0020368                0.0071385
```

`nodegr` is an indicator variable for whether the individual in the worker training program has a high school diploma. For such variables, the Kolmogorov-Smirnov test results are not

[10] No model is estimated unless multivariate tests are requested via the `mv` option.

presented because they are the equivalent to the results from t-tests.

The output labels the `nodegr` balance statistics as 'V8' because it was the eighth variable listed in the formula provided to `MatchBalance`. There are two columns for each variable. The first containing the pre-matching balance statistics and the second one the post-matching statistics.

Four different sets of balance statistics are provided for each variable. The first set consists of the means for the treatment and control groups. The second set contains summary statistics based on standardized empirical-QQ plots. The mean, median and maximum differences in the standardized empirical-QQ plots are provided. The third set of statistics consists of summary statistics from the raw empirical-QQ plots so they are on the scale of the variable in question. And the last set of statistics provides the variance ratio of treatment over control (which should equal 1 if there is perfect balance), and the t-test of difference of means (the paired t-test is provided post-matching). If they are calculated, the bootstrap Kolmogorov-Smirnov test results are also provided here.

The balance results make clear that `nodegr` is poorly balanced both before *and* after matching. Seventy-one percent of treatment observations have a high school diploma while seventy-seven percent of control observations do. And this difference is highly significant.

Next, let's consider another variable, `re74`, which is real earnings of participants in 1974.

```
***** (V9) re74 *****
                        Before Matching           After Matching
mean treatment........     2095.6                    2095.6
mean control..........     2107.0                    2193.3

mean std eQQ diff.....   0.019223                  0.054701
med  std eQQ diff.....   0.015800                  0.050872
max  std eQQ diff ....   0.047089                   0.12209

mean raw eQQ diff.....     487.98                    869.16
med  raw eQQ diff.....          0                         0
max  raw eQQ diff.....       8413                     10305

var ratio (Tr/Co).....     0.7381                   0.75054
T-test p-value........    0.98186                   0.84996
KS Bootstrap p-value..      0.587                 < 2.22e-16
KS Naive p-value......    0.97023                  0.011858
KS Statistic..........   0.047089                   0.12209
```

The balance of this variable has been made ***worse*** by matching. Before matching, treatment and control observations were only 11.4 dollars apart and this difference was not significant as judged by either a t-test for difference of means or by the Kolmogorov-Smirnov test which tests for a significant difference across the entire distribution. After matching, the mean difference increases to almost 100 dollars, but it still not significant. Unfortunately, the mean, median and maximum differences in the empirical-QQ plots increase sharply. And consistent with this, the KS tests shows a large and significant difference between the distribution of control and treatment observations.

Figure 1 plots the empirical-QQ plot of this variable before and after matching, and it shows that balance has been made worse by matching. The after matching portion of this figure (without the captions) was generated by the following code:

```
qqplot(re74[rr1$index.control], re74[rr1$index.treated])
abline(coef=c(0,1), col=2)
```

The `index.control` and `index.treated` indices which are in the object returned by `Match` are vectors containing the observation numbers from the original dataset for the treated (control) observations in the matched dataset. Both indices together can be used to construct the matched dataset. The matched dataset is also returned in the `mdata` object—see Appendix B for details.

This example shows that it is important to not simply look at differences of means. It is important to examine more general summaries of the distributions. Both the descriptive eQQ statistics and the KS test made clear that matching resulted in worse balance for this variable.

When faced with a propensity score which makes balance *worse*, it is sometimes possible to learn from the balance output and improve the propensity score. However, because the covariates are correlated with each other, it is difficult to know exactly how one should change the propensity score model. For example, the no highschool degree variable has significant imbalance both before and after matching. Should we interact it with other variables or do something else? It may be the case that we should not change the specification of `nodegr`, but instead change the specification of some other variable with which `nodegr` is correlated. In this example, that turns out to work.

Consider the following propensity score model proposed by Dehejia and Wahba (1999) to be used for the LaLonde data:

```
dw.pscore  <- glm(treat~age + I(age^2) + educ + I(educ^2) + black +
                  hisp + married + nodegr + re74  + I(re74^2) + re75 +
                  I(re75^2) + u74 + u75, family=binomial, data=lalonde)
```

This model adds second-order polynomials to the continuous variables we have: `age`, `educ`, `re74` and `re75`. And it adds indicator variables for whether income in 1974 and 1975 were zero: `u74`, `u75`. Note that this pscore model does not do anything different with `nodegr` than the previous one we used. The Dehejia and Wahba model does, however, perform significantly better.

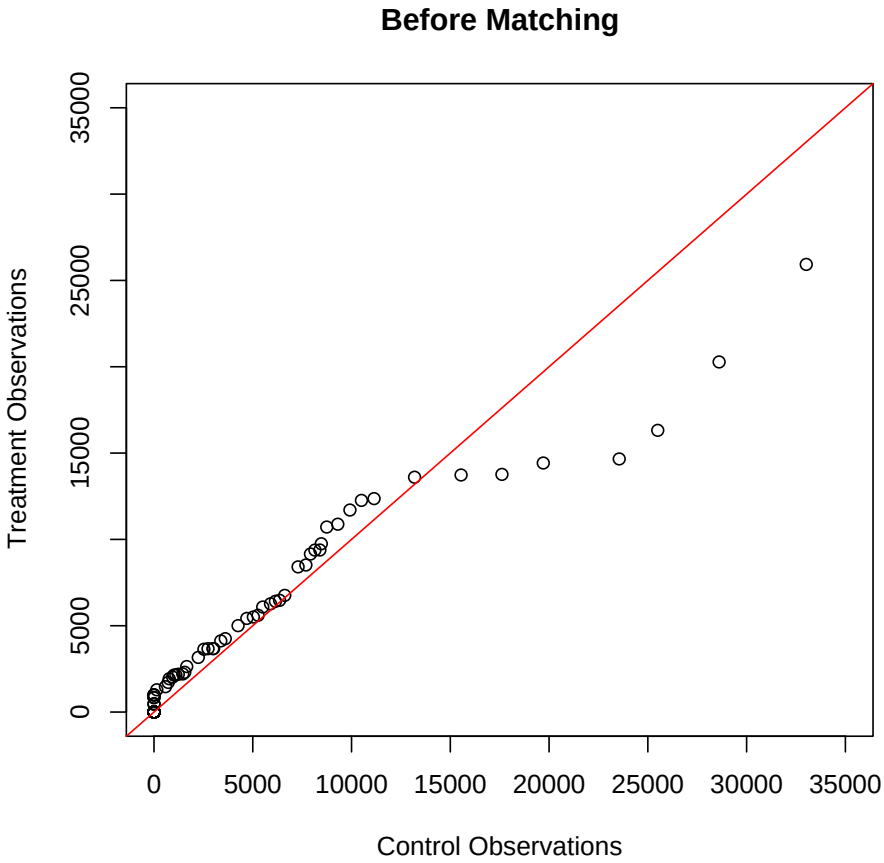
Consider the balance tests for the no highschool diploma variable:

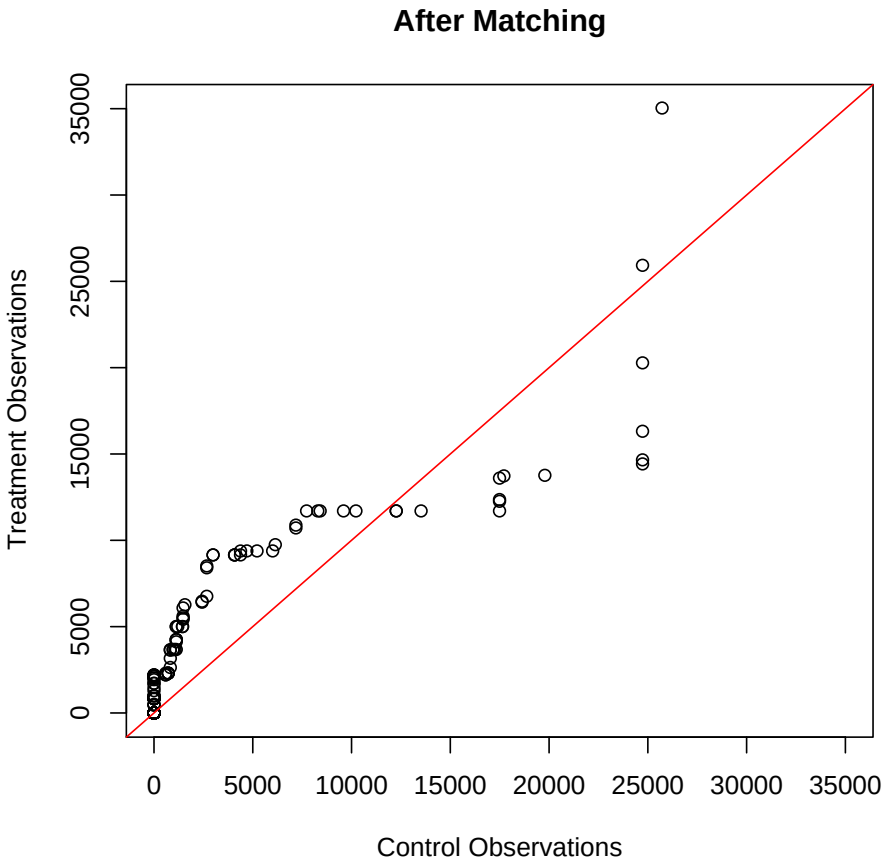
```
***** (V8) nodegr *****
                        Before Matching          After Matching
mean treatment........    0.70811                  0.70811
mean control..........    0.83462                  0.69189

mean std eQQ diff.....   0.063254                 0.0072254
med  std eQQ diff.....   0.063254                 0.0072254
max  std eQQ diff ....    0.12651                  0.014451

mean raw eQQ diff.....    0.12432                  0.014451
```

Figure 1: Empirical-QQ Plot of 're74' Before and After Pscore Matching





```
med  raw eQQ diff.....          0                          0
max  raw eQQ diff.....          1                          1

var ratio (Tr/Co).....     1.4998                    0.96957
T-test p-value........  0.0020368                    0.49161
```

The balance for this variable has now been significantly improved from that of the unmatched dataset. The difference has been shrunk to the point that the remaining imbalance in this covariate is probably not a serious concern.

The balance in the income in 1974 is better than that produced by the previous pscore model, but it is still worse than balance in the unmatched data:

```
***** (V9) re74 *****
                       Before Matching           After Matching
mean treatment........     2095.6                    2095.6
mean control..........     2107.0                    1624.3

mean std eQQ diff.....   0.019223                  0.019782
med  std eQQ diff.....   0.015800                  0.018786
max  std eQQ diff ....   0.047089                  0.046243

mean raw eQQ diff.....     487.98                    467.33
med  raw eQQ diff.....          0                         0
max  raw eQQ diff.....       8413                     12410

var ratio (Tr/Co).....     0.7381                    2.2663
T-test p-value........    0.98186                   0.22745
KS Bootstrap p-value..      0.569                     0.249
KS Naive p-value......    0.97023                    0.8532
KS Statistic..........   0.047089                  0.046243
```

The means of the variable across treatment and control groups and the standardized mean, median and maximum difference in the eQQ plots are *increased* by matching. Although the differences are not significant, they are if we look at the balance output for `re74`^{2}:

```
***** (V10) I(re74^2) *****
                       Before Matching           After Matching
mean treatment........   28141434                  28141434
mean control..........   36667413                  13117852

mean std eQQ diff.....   0.019223                  0.019782
med  std eQQ diff.....   0.015800                  0.018786
max  std eQQ diff ....   0.047089                  0.046243

mean raw eQQ diff.....   13311731                  10899373
med  raw eQQ diff.....          0                         0
max  raw eQQ diff.....  365146387                 616156569
```

```
var ratio (Tr/Co).....    0.50382                    7.9006
T-test p-value........    0.51322                   0.08604
KS Bootstrap p-value..      0.569                     0.249
KS Naive p-value......    0.97023                    0.8532
KS Statistic..........   0.047089                  0.046243
```

Note that the eQQ and KS test results are exactly the same for `re74` and `re74`^{2} as is to be expected because these non-parametric tests depend on the ranks of the observations rather than their precise values. However, the KS test is less sensitive to mean differences than the t-test test. It is more sensitive than the t-test to differences in the distributions beyond the first two moments. In this case, the t-test p-value for `re74`^{2} is much lower than it is for `re74`: 0.086 versus 0.23.

Since the previous outcome is usually the most important confounder we need to worry about, the remaining imbalance in this variable is of serious concern. And it is further troubling that matching is making balance worse in this variable than doing nothing at all!

As this example hopefully demonstrates, moving back and forth from balance statistics to changing the matching model is a tedious process. Fortunately, as described in Section 2.3, the problem can be clearly posed as an optimization problem which can be algorithmically solved.

3.2. Genetic Matching

The `GenMatch` function can be used for our example problem, and it greatly improves balance even over the Dehejia and Wahba (1999) propensity score model. `GenMatch` can be used with our without a pscore model. In this example, we will not make use of any pscore model just to demonstrate that `GenMatch` can perform well even without a human providing such a model. However, in general, inclusion of a good pscore model helps `GenMatch`.

```
X  <- cbind(age, educ, black, hisp, married, nodegr, re74, re75, u74, u75)

BalanceMatrix <- cbind(age, I(age^2), educ, I(educ^2), black,
                       hisp, married, nodegr, re74 , I(re74^2), re75,
                       I(re75^2), u74, u75, I(re74*re75), I(age*nodegr),
                       I(educ*re74), I(educ*re75))

gen1 <- GenMatch(Tr=Tr, X=X, BalanceMatrix=BalanceMatrix, pop.size=10000)
```

`GenMatch` takes four key arguments. The first two, `Tr` and `X`, are just the same as those of the `Match` function: the first is a vector for the treatment indicator and the second a matrix which contains the covariates which we wish to match on. The third key argument, `BalanceMatrix`, is a matrix containing the variables we wish to achieve balance on. This is by default equal to `X`, but it can be a matrix which contains more or less variables than `X` or variables which are transformed in various ways. It should generally contain the variables and the function of these variables that we wish to balance. In this example, I have made `BalanceMatrix` contain the same terms we had `MatchBalance` test balance for, and this, in general, is good

practice. If you care about balance for a given function of the covariates, you should put it in `BalanceMatrix` just like how you should put it into the equation in `MatchBalance`.

The `pop.size` argument is important and greatly influences how long the function takes to run. This argument controls the population size used by the evolutionary algorithm–i.e., it is the number of individuals `genoud` uses to solve the optimization problem. This argument is also the number of random trail solutions which are tried at the beginning of the search process. The theorems proving that genetic algorithms find good solutions are asymptotic in population size. Therefore, it is important that this value not be small (Vose 1993; Nix and Vose 1992). On the other hand, computational time is finite so obvious trade-offs must be made.

`GenMatch` has a large number of other options which are detailed in Appendix C. The options controlling features of the matching itself, such as whether to match with replacement, are the same as those of the `Match` function. But many other options are specific to `GenMatch` because they control the optimization process. The most important of these aside from `pop.size`, are `wait.generations` and `max.generations`.

In order to obtain balance statistics, we can simply do the following with the output object (`gen1`) returned by the call to `GenMatch` above:

```
mgen1 <- Match(Y=Y, Tr=Tr, X=X, Weight.matrix=gen1)

MatchBalance(treat~age + I(age^2) + educ + I(educ^2) + black +
             hisp + married + nodegr + re74  + I(re74^2) + re75 + I(re75^2) +
             u74 + u75 + I(re74*re75) + I(age*nodegr) + I(educ*re74) + I(educ*re75),
             data=lalonde, match.out=mgen1, nboots=1000)
```

The balance results from this `GenMatch` run are excellent. The *smallest* p-value across all of the variables tested in `MatchBalance` is 0.46 (for $\texttt{re75}^2$) compared with 0.086 for the DW pscore model (for $\texttt{re74}^2$) and the pre-matching value of 0.002 (for `nodegr`). Here is, for example, the balance output for `re74` and $\texttt{re74}^2$:

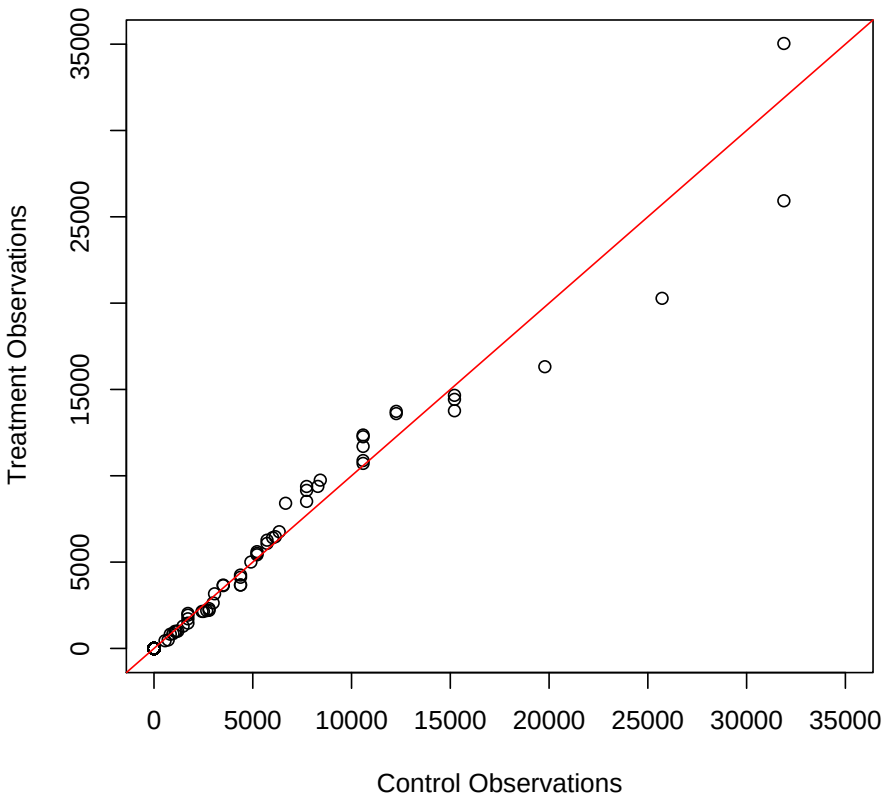
```
***** (V9) re74 *****
                       Before Matching           After Matching
mean treatment........     2095.6                    2095.6
mean control..........     2107.0                    2097.0

mean std eQQ diff.....   0.019223                 0.0068569
med  std eQQ diff.....   0.015800                 0.0075472
max  std eQQ diff ....   0.047089                  0.018868

mean raw eQQ diff.....     487.98                    174.65
med  raw eQQ diff.....          0                         0
max  raw eQQ diff.....       8413                    5956.7

var ratio (Tr/Co).....     0.7381                   0.94475
T-test p-value........    0.98186                   0.99006
KS Bootstrap p-value..      0.558                     0.988
KS Naive p-value......    0.97023                         1
```

Figure 2: Empirical-QQ Plot of 're74' Using GenMatch




```
KS Statistic..........   0.047089                  0.018868

***** (V10) I(re74^2) *****
                        Before Matching           After Matching
mean treatment........   28141434                  28141434
mean control..........   36667413                  29536169

mean std eQQ diff.....   0.019223                 0.0068569
med  std eQQ diff.....   0.015800                 0.0075472
max  std eQQ diff ....   0.047089                  0.018868

mean raw eQQ diff.....   13311731                   5301160
med  raw eQQ diff.....          0                         0
max  raw eQQ diff.....  365146387                 344393163

var ratio (Tr/Co).....    0.50382                   0.85573
T-test p-value........    0.51322                   0.69167
KS Bootstrap p-value..      0.558                     0.988
KS Naive p-value......    0.97023                         1
KS Statistic..........   0.047089                  0.018868
```

The balance for `re74` is now excellent and better than the pre-matching balance. The empirical-QQ plot for this variable, as shown in Figure 2, now looks rather good.

For the `nodegr` variable balance is also excellent:

```
***** (V8) nodegr *****
                       Before Matching          After Matching
mean treatment........    0.70811                  0.70811
mean control..........    0.83462                  0.71351

mean std eQQ diff.....   0.063254                 0.0018868
med  std eQQ diff.....   0.063254                 0.0018868
max  std eQQ diff ....    0.12651                 0.0037736

mean raw eQQ diff.....    0.12432                 0.0037736
med  raw eQQ diff.....          0                         0
max  raw eQQ diff.....          1                         1

var ratio (Tr/Co).....     1.4998                    1.0111
T-test p-value........  0.0020368                   0.56406
```

Now that we have achieved excellent balance, we can examine our estimate of the causal effect and its standard error. The code for this is:

```
summary(mgen1)
```

And the output which results is:

```
Estimate...  1649.8
AI SE......  870.99
T-stat.....  1.8941
p.val......  0.058208


Original number of observations..............  445
Original number of treated obs...............  185
Matched number of observations...............  185
Matched number of observations  (unweighted).  265
```

The causal estimate is $1,649.80 with a standard error of 870.99. By default, the Abadie-Imbens (AI) standard error is printed (Abadie and Imbens 2006). In order to also obtain the usual Neyman standard error, one may call the `summary` function with the `full=TRUE` option.

The `summary` function also provides the number of observations in total (445), the number of treated observations (185), the number of matched pairs that were produced when the ties are properly weighted (185), and the number of matched pairs without using the weights which adjust for ties (265).

3.3. Parallel and Cluster Processing

GenMatch is a computationally intensive algorithm because it constructs matched datasets for each trail set of covariate weights. Fortunately, as with most genetic algorithms, the algorithm easily parallelizes. This functionality has been built directly in the rgenoud package and be readily accessed by GenMatch. The parallelization can be used for either multiple CPU computers or a cluster of computers, and makes use of R's snow (Simple Network of Workstations) package. Simulations to estimate how well the parallel algorithm scales with multiple CPUs are provided below. On a single computer with multiple CPUs, the proportion of time saved is almost linear in the number of CPUs if the dataset size is large. For a cluster of separate computers, the algorithm is significantly faster for every extra node which is added, but the time savings are significantly less than linear. The exact amount of time saved depends on network latency and a host of other factors.

Two GenMatch options control the parallel processing: cluster and balance. The cluster option can either be an object of the 'cluster' class returned by one of the makeCluster commands in the snow package or a vector of machine names so that GenMatch can setup the cluster automatically via secure-shell (ssh). If it is the latter, the vector passed to the cluster option should look like the following:

```
c("localhost","localhost","musil","musil","deckard")
```

This vector would create a cluster with four nodes: two on the localhost another on 'deckard' and two on the machine named 'musil'. Two nodes on a given machine make sense if the machine has two or more chips/cores. GenMatch will setup a SOCK cluster by a call to makeSOCKcluster. This will require the user to type in her password for each node as the cluster is by default created via ssh. One can add on user names to the machine name if it differs from the current shell: username@musil. Other cluster types, such as PVM and MPI, which do not require passwords, can be created by directly calling makeCluster, and then passing the returned cluster object to GenMatch. For example, one can manually setup a cluster with a direct call to makeCluster as follows.

```
library(snow)
library(Matching)

#we could either create the connection here, outside of GenMatch, or
#let GenMatch do it.  But let's do it here so we don't have to keep
#setting it up.

cl <- makeCluster(c("musil","quetelet","quetelet"), type="SOCK")


data(lalonde)
attach(lalonde)

#The covariates we want to match on
X = cbind(age, educ, black, hisp, married, nodegr, u74, u75, re75, re74);

genout <- GenMatch(Tr=treat, X=X, cluster=cl)
```

```
#Manually stop the cluster because we manually started it.
stopCluster(cl)
```

The second `GenMatch` option which controls the behavior of parallel processing is the `balance` option. This is a logical flag which controls if load balancing is done across the cluster. Load balancing can result in better cluster utilization; however, increased communication can reduce performance. This options is best used if each individual call to `Match` takes at least several minutes to calculate or if the nodes in the cluster vary significantly in their performance.

Designing parallel software applications is difficult. A lot of work and trail-and-error has gone into writing the C++ functions which `GenMatch` relies upon to ensure that they are reliable and fast when run either serially or in parallel. Parallel execution is especially tricky because an algorithm which may be fast in serial mode can cause unexpected bottlenecks when run in parallel (such as a cache-bottleneck when executing SSE3 instructions via BLAS).

We now explore how well `GenMatch` scales with additional CPUs by using the following benchmark code:

```
library(Matching)
data(lalonde)
attach(lalonde)

X = cbind(age, educ, black, hisp, married, nodegr, u74, u75, re75, re74);
Xbig <- rbind(X, X, X, X)

Ybig <- c(treat, treat, treat, treat)

GenMatch(Tr=Ybig, X=Xbig, BalanceMatrix=Xbig, estimand="ATE", M=1,
         pop.size=1000, max.generations=10, wait.generations=1,
         int.seed=3818, unif.seed=3527,
         cluster=c("localhost","localhost",
                   "localhost","localhost"))))
```

This example makes use of four computer chips: note the four calls to `localhost`. The dataset is replicated four times (e.g., `Xbig` and `Ybig`) to obtain 1780 observations. And smaller datasets are created by not replicating the observations as often. Table 1 presents the average run times of this code as it is run on one to four CPUs and on various dataset sizes.[11]

`GenMatch` scales more efficiently across computer chips as the dataset size becomes larger. With 1780 observations, using four computer chips results in a run time which is 29% of the single chip run time. This is a good increase in performance given that if parallelization were as efficient as possible, using four chips would result in 25% of the run time as that of using a single chip. Of course, perfect parallelization is not possible given the overhead involved in

[11]There is no significant difference in run times across different invocations of the same commands so no variance estimates are presented, just average run times. A four core Xeon processor (5150 @ 2.66GHz) computer running 64-bit Linux (Ubuntu Dapper) was used, and all extraneous daemons were shutdown.

Table 1: Using Multiple Computer Chips to Run `GenMatch`

	1 CPU	2 CPUs	3 CPUs	4 CPUs
1780 Observations				
run time (seconds)	2557	1372	950	749
x CPU/1 CPU run time		0.54	0.37	0.29
1335 Observations				
run time (seconds)	826	475	317	255
x CPU/1 CPU run time		0.58	0.38	0.31
890 Observations				
run time (seconds)	532	338	233	193
x CPU/1 CPU run time		0.64	0.44	0.36

setting up parallel computations.[12] Note that that scaling from one to two CPUs is closer to the theoretical efficiency bound ($1.08 = \frac{.54}{.5}$) then scaling from one to four chips ($1.16 = \frac{.29}{25}$). This may be due to the issue pointed out in footnote 12.

With 890 observations, using four CPUs takes 36% of the run time as only using one CPU. This is a significantly smaller efficiency gain than that achieved with the dataset with 1780 observations.

It is clear from Table 1 that the computational time it takes to execute a matching algorithm does not increase linearly with sample size. The computational time increases as a polynomial of the sample size, and the average asymptotic order of the `Match` function is approximately $O(N^2)log(N)$.[13] The run times in Table 1 are generally consistent with this. Although the `Match` function increases in polynomial time, the problem which `GenMatch` attempts to solve (that of finding the best distance metric) increases exponentially in sample size, just like the traveling salesman problem. That is, the set of possible matched datasets grows exponentially with sample size.

4. Conclusion

The functions in `Matching` have many more options than can be reviewed in this brief paper. For additional details the R help files for the three main functions are included in the appendices: `Match`, `GenMatch`, and `MatchBalance`. The `Matching` package includes five additional functions: `Matchby` (for large datasets), `qqstats` (descriptive eQQ statistics), `ks.boot`

[12]The efficiency of this parallelization is more impressive given that the test runs were run on a four core Xeon processor (5150) which is not really a four core chip. This chip actually consists of two (dual-core) Woodcrest chips. The two chips have no way to communicate directly with each other. All communications between them have to go through a shared front-side bus with the memory controller hub, or north bridge. And each chip independently accesses the cache coherency scheme. See http://www.techreport.com/reviews/2006q4/xeon-vs-opteron/index.x?pg=3.

[13]The precise asymptotic order is difficult to calculate because assumptions have to be made about various features of the data such as the proportion of ties.

(bootstrap version of `ks.test`), `balanceUV` (univariate balance statistics), and `balanceMV` (multivariate balance statistics). The help files for these functions may be found in the package itself.

A great deal of effort has been made in order to ensure that the matching functions are as fast as possible. The computationally intensive functions are written in C++ which make extensive use of the BLAS libraries, and `GenMatch` can be used with multiple computers, CPUs or cores to perform parallel computations. The C++ functions have been written so that the GNU g++ compiler does a good job of optimizing them. Indeed, compiling the `Matching` package with the Intel C++ compiler does not result in faster code. This is unusual with floating point code, and is the result of carefully writing code so that the GNU compiler is able to optimize it aggressively.

After intensive benchmarking and instrumenting the code, it was determined that performance on OS X was seriously limited because the default OS X memory allocator is not as efficient as Doug Lea's malloc given the frequent memory allocations made by the matching code. The matching algorithm was rewritten in order to be more efficient with memory on all platforms, and on OS X, `Matching` is compiled against Lea's malloc which is something more packages for R may wish to do. For details see Sekhon (2006b).

The literature on matching methods is developing quickly with new innovations being made by a variety of researchers in fields ranging from economics, epidemiology and political science to sociology and statistics. Hence, new options are being added frequently.

A. Equal Percent Bias Reduction (EPBR)

Affinely invariant matching methods, such as Mahalanobis metric matching and propensity score matching (if the propensity score is estimated by logistic regression), are equal percent bias reducing if all of the covariates used have ellipsoidal distributions (Rubin and Thomas 1992)—e.g., distributions such as the normal or t—or if the covariates are mixtures of proportional ellipsoidally symmetric (DMPES) distributions (Rubin and Stuart 2006).[14]

To formally define EPBR, let Z be the expected value of X in the matched control group. Then, as outlined in Rubin (1976a), a matching procedure is EPBR if

$$E(X \mid T = 1) - Z \quad = \gamma \left\{ E(X \mid T = 1) - E(X \mid T = 0) \right\}$$

for a scalar $0 \leq \gamma \leq 1$. In other words, we say that a matching method is EPBR for X when the percent reduction in the biases of each of the matching variables is the same. One obtains the same percent reduction in bias for any linear function of X if and only if the matching method is EPBR for X. Moreover, if a matching method is not EPBR for X, the bias for some linear function of X is increased even if all univariate covariate means are closer in the matched data than the unmatched (Rubin 1976a).

Even if the covariates have elliptic distributions, in finite samples they may not. Then Mahalanobis distance may not be optimal because the matrix used to scale the distances, the covariance matrix of X, can be improved upon.

[14]Note that DMPES defines a limited set of mixtures. In particular, countably infinite mixtures of ellipsoidal distributions where: (1) all inner products are proportional and (2) where the centers of each constituent ellipsoidal distribution are such that all best linear discriminants between any two components are also proportional.

The EPBR property itself is limited and in a given substantive problem it may not be desirable. This can arise if it is known based on theory that one covariate has a large nonlinear relationship with the outcome while another does not—e.g., $Y = X_1^4 + X_2$, where $X_1 > 1$. In such a case, reducing bias in X_1 will be more important than X_2.

B. Match: Multivariate and Propensity Score Matching

`Match`	*Multivariate and Propensity Score Matching Estimator for Causal Inference*

Description

`Match` implements a variety of algorithms for multivariate matching including propensity score, Mahalanobis and inverse variance matching. The function is intended to be used in conjunction with the `MatchBalance` function which determines the extent to which `Match` has been able to achieve covariate balance. In order to do propensity score matching, one should estimate the propensity model before calling `Match`, and then send `Match` the propensity score to use. `Match` enables a wide variety of matching options including matching with or without replacement, bias adjustment, different methods for handling ties, exact and caliper matching, and a method for the user to fine tune the matches via a general restriction matrix. Variance estimators include the usual Neyman standard errors, Abadie-Imbens standard errors, and robust variances which do not assume a homogeneous causal effect. The `GenMatch` function can be used to *automatically find balance* via a genetic search algorithm which determines the optimal weight to give each covariate.

Usage

```
Match(Y=NULL, Tr, X, Z = X, V = rep(1, length(Y)), estimand = "ATT", M = 1,
      BiasAdjust = FALSE, exact = NULL, caliper = NULL, replace=TRUE, ties=TRUE,
      CommonSupport=FALSE,Weight = 1, Weight.matrix = NULL, weights = NULL,
      Var.calc = 0, sample = FALSE, tolerance = 1e-05,
      distance.tolerance = 1e-05, restrict=NULL,
      match.out = NULL, version="standard")
```

Arguments

`Y`	A vector containing the outcome of interest. Missing values are not allowed. An outcome vector is not required because the matches generated will be the same regardless of the outcomes. Of course, without any outcomes no causal effect estimates will be produced, only a matched dataset.

`Tr` A vector indicating the observations which are in the treatment regime and those which are not. This can either be a logical vector or a real vector where 0 denotes control and 1 denotes treatment.

`X` A matrix containing the variables we wish to match on. This matrix may contain the actual observed covariates or the propensity score or a combination of both. All columns of this matrix must have positive variance or `Match` will return an error.

`Z` A matrix containing the covariates for which we wish to make bias adjustments.

`V` A matrix containing the covariates for which the variance of the causal effect may vary. Also see the `Var.calc` option, which takes precedence.

`estimand` A character string for the estimand. The default estimand is "ATT", the sample average treatment effect for the treated. "ATE" is the sample average treatment effect, and "ATC" is the sample average treatment effect for the controls.

`M` A scalar for the number of matches which should be found. The default is one-to-one matching. Also see the `ties` option.

`BiasAdjust` A logical scalar for whether regression adjustment should be used. See the `Z` matrix.

`exact` A logical scalar or vector for whether exact matching should be done. If a logical scalar is provided, that logical value is applied to all covariates in `X`. If a logical vector is provided, a logical value should be provided for each covariate in `X`. Using a logical vector allows the user to specify exact matching for some but not other variables. When exact matches are not found, observations are dropped. `distance.tolerance` determines what is considered to be an exact match. The `exact` option takes precedence over the `caliper` option.

`caliper` A scalar or vector denoting the caliper(s) which should be used when matching. A caliper is the distance which is acceptable for any match. Observations which are outside of the caliper are dropped. If a scalar caliper is provided, this caliper is used for all covariates in `X`. If a vector of calipers is provided, a caliper value should be provided for each covariate in `X`. The caliper is interpreted to be in standardized units. For example, `caliper=.25` means that all matches not equal to or within .25 standard deviations of each covariate in `X` are dropped. Note that dropping observations generally changes the quantity being estimated.

`replace` A logical flag for whether matching should be done with replacement. Note that if `FALSE`, the order of matches generally matters. Matches will be found in the same order as the data are sorted. Thus, the match(es) for the first observation will be found first, the match(es) for the second observation will be found second, etc. Matching without replacement will generally increase bias. Ties are randomly broken when `replace==FALSE` —see the `ties` option for details.

`ties` A logical flag for whether ties should be handled deterministically. By default `ties==TRUE`. If, for example, one treated observation matches more

than one control observation, the matched dataset will include the multiple matched control observations and the matched data will be weighted to reflect the multiple matches. The sum of the weighted observations will still equal the original number of observations. If `ties==FALSE`, ties will be randomly broken. *If the dataset is large and there are many ties, setting `ties=FALSE` often results in a large speedup.* Whether two potential matches are close enough to be considered tied, is controlled by the `distance.tolerance` option.

`CommonSupport`

This logical flag implements the usual procedure by which observations outside of the common support of a variable (usually the propensity score) across treatment and control groups are discarded. The `caliper` option is to be preferred to this option because `CommonSupport`, consistent with the literature, only drops *outliers* and leaves *inliers* while the caliper option drops both. If `CommonSupport==TRUE`, common support will be enforced on the first variable in the `X` matrix. Note that dropping observations generally changes the quantity being estimated. Use of this option renders it impossible to use the returned objects `index.treated` and `index.control` to reconstruct the matched dataset. The returned object `mdata` will, however, still contain the matched dataset. Seriously, don't use this option; use the `caliper` option instead.

`Weight`

A scalar for the type of weighting scheme the matching algorithm should use when weighting each of the covariates in `X`. The default value of 1 denotes that weights are equal to the inverse of the variances. 2 denotes the Mahalanobis distance metric, and 3 denotes that the user will supply a weight matrix (`Weight.matrix`). Note that if the user supplies a `Weight.matrix`, `Weight` will be automatically set to be equal to 3.

`Weight.matrix`

This matrix denotes the weights the matching algorithm uses when weighting each of the covariates in `X`—see the `Weight` option. This square matrix should have as many columns as the number of columns of the `X` matrix. This matrix is usually provided by a call to the `GenMatch` function which finds the optimal weight each variable should be given so as to achieve balance on the covariates.

For most uses, this matrix has zeros in the off-diagonal cells. This matrix can be used to weight some variables more than others. For example, if `X` contains three variables and we want to match as best as we can on the first, the following would work well:

```
> Weight.matrix <- diag(3)
> Weight.matrix[1,1] <- 1000/var(X[,1])
> Weight.matrix[2,2] <- 1/var(X[,2])
> Weight.matrix[3,3] <- 1/var(X[,3])
```

This code changes the weights implied by the inverse of the variances by multiplying the first variable by a 1000 so that it is highly weighted. In order to enforce exact matching see the `exact` and `caliper` options.

`weights` A vector the same length as `Y` which provides observation specific weights.

`Var.calc` A scalar for the variance estimate that should be used. By default `Var.calc=0` which means that homoscedasticity is assumed. For values of `Var.calc > 0`, robust variances are calculated using `Var.calc` matches.

`sample` A logical flag for whether the population or sample variance is returned.

`tolerance` This is a scalar which is used to determine numerical tolerances. This option is used by numerical routines such as those used to determine if a matrix is singular.

`distance.tolerance`
This is a scalar which is used to determine if distances between two observations are different from zero. Values less than `distance.tolerance` are deemed to be equal to zero. This option can be used to perform a type of optimal matching

`restrict` A matrix which restricts the possible matches. This matrix has one row for each restriction and three columns. The first two columns contain the two observation numbers which are to be restricted (for example 4 and 20), and the third column is the restriction imposed on the observation-pair. Negative numbers in the third column imply that the two observations cannot be matched under any circumstances, and positive numbers are passed on as the distance between the two observations for the matching algorithm. The most commonly used positive restriction is `0` which implies that the two observations will always be matched.

Exclusion restrictions are even more common. For example, if we want to exclude the observation pair 4 and 20 and the pair 6 and 55 from being matched, the restrict matrix would be: `restrict=rbind(c(4,20,-1),c(6,55,-1))`

`match.out` The return object from a previous call to `Match`. If this object is provided, then `Match` will use the matches found by the previous invocation of the function. Hence, `Match` will run faster. This is useful when the treatment does not vary across calls to `Match` and one wants to use the same set of matches as found before. This often occurs when one is trying to estimate the causal effect of the same treatment (`Tr`) on different outcomes (`Y`). When using this option, be careful to use the same arguments as used for the previous invocation of `Match` unless you know exactly what you are doing.

`version` The version of the code to be used. The "fast" C/C++ version of the code does not calculate Abadie-Imbens standard errors. Additional speed can be obtained by setting `ties=FALSE` or `replace=FALSE` if the dataset is large and/or has many ties. The "legacy" version of the code does not make a call to an optimized C/C++ library and is included only for historical compatibility. The "fast" version of the code is significantly faster than the "standard" version for large datasets, and the "legacy" version is much slower than either of the other two.

Details

This function is intended to be used in conjunction with the `MatchBalance` function which checks if the results of this function have actually achieved balance. The results of this function can be summarized by a call to the `summary.Match` function. If one wants to do propensity score matching, one should estimate the propensity model before calling `Match`, and then place the fitted values in the `X` matrix—see the provided example.

The `GenMatch` function can be used to *automatically find balance* by the use of a genetic search algorithm which determines the optimal weight to give each covariate. The object returned by `GenMatch` can be supplied to the `Weight.matrix` option of `Match` to obtain estimates.

`Match` is often much faster with large datasets if `ties=FALSE` or `replace=FALSE`—i.e., if matching is done by randomly breaking ties or without replacement. Also see the `Matchby` function. It provides a wrapper for `Match` which is much faster for large datasets when it can be used.

Three demos are included: `GerberGreenImai`, `DehejiaWahba`, and `AbadieImbens`. These can be run by calling the `demo` function such as by `demo(DehejiaWahba)`.

Value

`est` The estimated average causal effect.

`se` The Abadie-Imbens standard error. This standard error is principled if `X` consists of either covariates or a known propensity score because it takes into account the uncertainty of the matching procedure. If an estimated propensity score is used, the uncertainty involved in its estimation is not accounted for although the uncertainty of the matching procedure itself still is.

`est.noadj` The estimated average causal effect without any `BiasAdjust`. If `BiasAdjust` is not requested, this is the same as `est`.

`se.standard` The usual standard error. This is the standard error calculated on the matched data using the usual method of calculating the difference of means (between treated and control) weighted by the observation weights provided by `weights`. Note that the standard error provided by `se` takes into account the uncertainty of the matching procedure while `se.standard` does not. Neither `se` nor `se.standard` take into account the uncertainty of estimating a propensity score. `se.standard` does not take into account any `BiasAdjust`. Summary of both types of standard error results can be requested by setting the `full=TRUE` flag when using the `summary.Match` function on the object returned by `Match`.

`se.cond` The conditional standard error. The practitioner should not generally use this.

`mdata` A list which contains the matched datasets produced by `Match`. Three datasets are included in this list: `Y`, `Tr` and `X`.

`index.treated`
A vector containing the observation numbers from the original dataset for the treated observations in the matched dataset. This index in conjunction with `index.control` can be used to recover the matched dataset produced by `Match`. For example, the `X` matrix used by `Match` can be recovered by `rbind(X[index.treated,],X[index.control,])`. The user should generally just examine the output of `mdata`.

`index.control`
A vector containing the observation numbers from the original data for the control observations in the matched data. This index in conjunction with `index.treated` can be used to recover the matched dataset produced by `Match`. For example, the `X` matrix used by `Match` can be recovered by `rbind(X[index.treated,],X[index.control,])`. The user should generally just examine the output of `mdata`.

`index.dropped`
A vector containing the observation numbers from the original data which were dropped (if any) in the matched dataset because of various options such as `caliper` and `exact`. If no observations were dropped, this index will be `NULL`.

`weights` The weight for the matched dataset. If all of the observations had a weight of 1 on input, they will have a weight of 1 on output if each observation was only matched once.

`orig.nobs` The original number of observations in the dataset.

`orig.wnobs` The original number of weighted observations in the dataset.

`orig.treated.nobs`
The original number of treated observations (unweighted).

`nobs` The number of observations in the matched dataset.

`wnobs` The number of weighted observations in the matched dataset.

`caliper` The `caliper` which was used.

`ecaliper` The size of the enforced caliper on the scale of the `X` variables. This object has the same length as the number of covariates in `X`.

`exact` The value of the `exact` function argument.

`ndrops` The number of weighted observations which were dropped either because of caliper or exact matching. This number, unlike `ndrops.matches`, takes into account observation specific weights which the user may have provided via the `weights` argument.

`ndrops.matches`
The number of matches which were dropped either because of caliper or exact matching.

Author(s)

Jasjeet S. Sekhon, UC Berkeley, sekhon@berkeley.edu, http://sekhon.berkeley.edu/.

References

Sekhon, Jasjeet S. 2007. "Multivariate and Propensity Score Matching Software with Automated Balance Optimization." *Journal of Statistical Software*. http://sekhon.berkeley.edu/papers/MatchingJSS.pdf

Sekhon, Jasjeet S. 2006. "Alternative Balance Metrics for Bias Reduction in Matching Methods for Causal Inference." Working Paper. http://sekhon.berkeley.edu/papers/SekhonBalanceMetrics.pdf

Abadie, Alberto and Guido Imbens. 2006. "Large Sample Properties of Matching Estimators for Average Treatment Effects." *Econometrica* 74(1): 235-267. http://ksghome.harvard.edu/~.aabadie.academic.ksg/sme.pdf

Diamond, Alexis and Jasjeet S. Sekhon. 2005. "Genetic Matching for Estimating Causal Effects: A General Multivariate Matching Method for Achieving Balance in Observational Studies." Working Paper. http://sekhon.berkeley.edu/papers/GenMatch.pdf

Imbens, Guido. 2004. Matching Software for Matlab and Stata. http://elsa.berkeley.edu/~imbens/estimators.shtml

See Also

Also see summary.Match, GenMatch, MatchBalance, Matchby, balanceMV, balanceUV, qqstats, ks.boot, GerberGreenImai, lalonde

Examples

```
#
# Replication of Dehejia and Wahba psid3 model
#
# Dehejia, Rajeev and Sadek Wahba. 1999.``Causal Effects in Non-Experimental Studies:
# Re-Evaluating the Evaluation of Training Programs.''Journal of the American
# Statistical Association 94 (448): 1053-1062.
#
data(lalonde)

#
# Estimate the propensity model
#
glm1  <- glm(treat~age + I(age^2) + educ + I(educ^2) + black +
             hisp + married + nodegr + re74  + I(re74^2) + re75 + I(re75^2) +
             u74 + u75, family=binomial, data=lalonde)
```

```
#
#save data objects
#
X  <- glm1$fitted
Y  <- lalonde$re78
Tr  <- lalonde$treat

#
# one-to-one matching with replacement (the "M=1" option).
# Estimating the treatment effect on the treated (the "estimand" option defaults to ATT).
#
rr  <- Match(Y=Y, Tr=Tr, X=X, M=1);
summary(rr)

#
# Let's check for balance
# 'nboots' and 'nmc' are set to small values in the interest of speed.
# Please increase to at least 500 each for publication quality p-values.
mb  <- MatchBalance(treat~age + I(age^2) + educ + I(educ^2) + black +
                    hisp + married + nodegr + re74  + I(re74^2) + re75 + I(re75^2) +
                    u74 + u75, data=lalonde, match.out=rr, nboots=10, nmc=10)
```

C. GenMatch: Genetic Matching

`GenMatch` *Genetic Matching*

Description

This function finds optimal balance using multivariate matching where a genetic search algorithm determines the weight each covariate is given. Balance is determined by examining cumulative probability distribution functions of a variety of standardized statistics. By default, these statistics include t-tests and Kolmogorov-Smirnov tests. A variety of descriptive statistics based on empirical-QQ (eQQ) plots can also be used or any user provided measure of balance. The statistics are not used to conduct formal hypothesis tests, because no measure of balance is a monotonic function of bias and because balance should be maximized without limit. The object returned by `GenMatch` can be supplied to the `Match` function (via the `Weight.matrix` option) to obtain causal estimates. `GenMatch` uses `genoud` to perform the genetic search. Using the `cluster` option, one may use multiple computers, CPUs or cores to perform parallel computations.

Usage

```
GenMatch(Tr, X, BalanceMatrix=X, estimand="ATT", M=1, weights=NULL,
         pop.size = 100, max.generations=100,
         wait.generations=4, hard.generation.limit=FALSE,
         starting.values=rep(1,ncol(X)),
         fit.func="pvals",
         MemoryMatrix=TRUE,
         exact=NULL, caliper=NULL, replace=TRUE, ties=TRUE,
         CommonSupport=FALSE, nboots=0, ks=TRUE, verbose=FALSE,
         tolerance = 1e-05,
         distance.tolerance=tolerance,
         min.weight=0, max.weight=1000,
         Domains=NULL, print.level=2,
         project.path=NULL,
         paired=TRUE, loss=1,
         data.type.integer=FALSE,
         restrict=NULL,
         cluster=FALSE, balance=TRUE, ...)
```

Arguments

`Tr` A vector indicating the observations which are in the treatment regime and those which are not. This can either be a logical vector or a real vector where 0 denotes control and 1 denotes treatment.

`X` A matrix containing the variables we wish to match on. This matrix may contain the actual observed covariates or the propensity score or a combination of both.

`BalanceMatrix` A matrix containing the variables we wish to achieve balance on. This is by default equal to `X`, but it can in principle be a matrix which contains more or less variables than `X` or variables which are transformed in various ways. See the examples.

`estimand` A character string for the estimand. The default estimand is "ATT", the sample average treatment effect for the treated. "ATE" is the sample average treatment effect, and "ATC" is the sample average treatment effect for the controls.

`M` A scalar for the number of matches which should be found. The default is one-to-one matching. Also see the `ties` option.

`weights` A vector the same length as `Y` which provides observation specific weights.

`pop.size` Population Size. This is the number of individuals `genoud` uses to solve the optimization problem. The theorems proving that genetic algorithms find good solutions are asymptotic in population size. Therefore, it is important that this value not be small. See `genoud` for more details.

`max.generations` Maximum Generations. This is the maximum number of generations that `genoud` will run when optimizing. This is a *soft* limit. The maximum generation limit will be binding only if `hard.generation.limit` has been

set equal to *TRUE*. Otherwise, `wait.generations` controls when optimization stops. See `genoud` for more details.

`wait.generations`
: If there is no improvement in the objective function in this number of generations, optimization will stop. The other options controlling termination are `max.generations` and `hard.generation.limit`.

`hard.generation.limit`
: This logical variable determines if the `max.generations` variable is a binding constraint. If `hard.generation.limit` is *FALSE*, then the algorithm may exceed the `max.generations` count if the objective function has improved within a given number of generations (determined by `wait.generations`).

`starting.values`
: This vector's length is equal to the number of variables in `X`. This vector contains the starting weights each of the variables is given. The `starting.values` vector is a way for the user to insert *one* individual into the starting population. `genoud` will randomly create the other individuals. These values correspond to the diagonal of the `Weight.matrix` as described in detail in the `Match` function.

`fit.func`
: The balance metric `GenMatch` should optimize. The user may choose from the following or provide a function:
`pvals`: maximize the p.values from (paired) t-tests and Kolmogorov-Smirnov tests conducted for each column in `BalanceMatrix`. Lexical optimization is conducted—see the `loss` option for details.
`qqmean.mean`: calculate the mean standardized difference in the eQQ plot for each variable. Minimize the mean of these differences across variables.
`qqmean.max`: calculate the mean standardized difference in the eQQ plot for each variable. Minimize the maximum of these differences across variables. Lexical optimization is conducted.
`qqmedian.mean`: calculate the median standardized difference in the eQQ plot for each variable. Minimize the median of these differences across variables.
`qqmedian.max`: calculate the median standardized difference in the eQQ plot for each variable. Minimize the maximum of these differences across variables. Lexical optimization is conducted.
`qqmax.mean`: calculate the maximum standardized difference in the eQQ plot for each variable. Minimize the mean of these differences across variables.
`qqmax.max`: calculate the maximum standardized difference in the eQQ plot for each variable. Minimize the maximum of these differences across variables. Lexical optimization is conducted.
The user may provide their own `fit.func`. This function needs to return a fit value that will be minimized. The function should expect two arguments. The first being the `matches` object returned by `GenMatch`—see below. And the second being a matrix which contains the variables to be balanced—i.e., the `BalanceMatrix` the user provided to `GenMatch`.

`MemoryMatrix` This variable controls if `genoud` sets up a memory matrix. Such a matrix ensures that `genoud` will request the fitness evaluation of a given set of parameters only once. The variable may be *TRUE* or *FALSE*. If it is *FALSE*, `genoud` will be aggressive in conserving memory. The most significant negative implication of this variable being set to *FALSE* is that `genoud` will no longer maintain a memory matrix of all evaluated individuals. Therefore, `genoud` may request evaluations which it has previously requested. When the number variables in `X` is large, the memory matrix consumes a large amount of RAM.

`genoud`'s memory matrix will require *significantly* less memory if the user sets `hard.generation.limit` equal to *TRUE*. Doing this is a good way of conserving memory while still making use of the memory matrix structure.

`exact` A logical scalar or vector for whether exact matching should be done. If a logical scalar is provided, that logical value is applied to all covariates in `X`. If a logical vector is provided, a logical value should be provided for each covariate in `X`. Using a logical vector allows the user to specify exact matching for some but not other variables. When exact matches are not found, observations are dropped. `distance.tolerance` determines what is considered to be an exact match. The `exact` option takes precedence over the `caliper` option. Obviously, if `exact` matching is done using *all* of the covariates, one should not be using `GenMatch` unless the `distance.tolerance` has been set unusually high.

`caliper` A scalar or vector denoting the caliper(s) which should be used when matching. A caliper is the distance which is acceptable for any match. Observations which are outside of the caliper are dropped. If a scalar caliper is provided, this caliper is used for all covariates in `X`. If a vector of calipers is provided, a caliper value should be provided for each covariate in `X`. The caliper is interpreted to be in standardized units. For example, `caliper=.25` means that all matches not equal to or within .25 standard deviations of each covariate in `X` are dropped. The `ecaliper` object which is returned by `GenMatch` shows the enforced caliper on the scale of the `X` variables. Note that dropping observations generally changes the quantity being estimated.

`replace` A logical flag for whether matching should be done with replacement. Note that if `FALSE`, the order of matches generally matters. Matches will be found in the same order as the data are sorted. Thus, the match(es) for the first observation will be found first, the match(es) for the second observation will be found second, etc. Matching without replacement will generally increase bias. Ties are randomly broken when `replace==FALSE`—see the `ties` option for details.

`ties` A logical flag for whether ties should be handled deterministically. By default `ties==TRUE`. If, for example, one treated observation matches more than one control observation, the matched dataset will include the multiple matched control observations and the matched data will be weighted to reflect the multiple matches. The sum of the weighted observations

will still equal the original number of observations. If `ties==FALSE`, ties will be randomly broken. *If the dataset is large and there are many ties, setting `ties=FALSE` often results in a large speedup.* Whether two potential matches are close enough to be considered tied, is controlled by the `distance.tolerance` option.

`CommonSupport`
: This logical flag implements the usual procedure by which observations outside of the common support of a variable (usually the propensity score) across treatment and control groups are discarded. The `caliper` option is to be preferred to this option because `CommonSupport`, consistent with the literature, only drops *outliers* and leaves *inliers* while the caliper option drops both. If `CommonSupport==TRUE`, common support will be enforced on the first variable in the `X` matrix. Note that dropping observations generally changes the quantity being estimated. Use of this option renders it impossible to use the returned object `matches` to reconstruct the matched dataset. Seriously, don't use this option; use the `caliper` option instead.

`nboots`
: The number of bootstrap samples to be run for the `ks` test. By default this option is set to zero so no bootstraps are done. See `ks.boot` for additional details.

`ks`
: A logical flag for if the univariate bootstrap Kolmogorov-Smirnov (KS) test should be calculated. If the ks option is set to true, the univariate KS test is calculated for all non-dichotomous variables. The bootstrap KS test is consistent even for non-continuous variables. By default, the bootstrap KS test is not used. To change this see the `nboots` option. If a given variable is dichotomous, a t-test is used even if the KS test is requested. See `ks.boot` for additional details.

`verbose`
: A logical flag for whether details of each fitness evaluation should be printed. Verbose is set to FALSE if the `cluster` option is used.

`tolerance`
: This is a scalar which is used to determine numerical tolerances. This option is used by numerical routines such as those used to determine if a matrix is singular.

`distance.tolerance`
: This is a scalar which is used to determine if distances between two observations are different from zero. Values less than `distance.tolerance` are deemed to be equal to zero. This option can be used to perform a type of optimal matching.

`min.weight`
: This is the minimum weight any variable may be given.

`max.weight`
: This is the maximum weight any variable may be given.

`Domains`
: This is a `ncol(X)` $\times 2$ matrix. The first column is the lower bound, and the second column is the upper bound for each variable over which `genoud` will search for weights. If the user does not provide this matrix, the bounds for each variable will be determined by the `min.weight` and `max.weight` options.

`print.level` This option controls the level of printing. There are four possible levels: 0 (minimal printing), 1 (normal), 2 (detailed), and 3 (debug). If level 2 is selected, `GenMatch` will print details about the population at each generation, including the best individual found so far. If debug level printing is requested, details of the `genoud` population are printed in the "genoud.pro" file which is located in the temporary `R` directory returned by the `tempdir` function. See the `project.path` option for more details. Because `GenMatch` runs may take a long time, it is important for the user to receive feedback. Hence, print level 2 has been set as the default.

`project.path` This is the path of the `genoud` project file. By default no file is produced unless `print.level=3`. In that case, `genoud` places its output in a file called "genoud.pro" located in the temporary directory provided by `tempdir`. If a file path is provided to the `project.path` option, a file will be created regardless of the `print.level`. The behavior of the project file, however, will depend on the `print.level` chosen. If the `print.level` variable is set to 1, then the project file is rewritten after each generation. Therefore, only the currently fully completed generation is included in the file. If the `print.level` variable is set to 2 or higher, then each new generation is simply appended to the project file. No project file is generated for `print.level=0`.

`paired` A flag for whether the paired `t.test` should be used when determining balance.

`loss` The loss function to be optimized. The default value, `1`, implies "lexical" optimization: all of the balance statistics will be sorted from the most discrepant to the least and weights will be picked which minimize the maximum discrepancy. If multiple sets of weights result in the same maximum discrepancy, then the second largest discrepancy is examined to choose the best weights. The processes continues iteratively until ties are broken.

If the value of `2` is used, then only the maximum discrepancy is examined. This was the default behavior prior to version 1.0. The user may also pass in any function she desires. Note that the option 1 corresponds to the `sort` function and option 2 to the `min` function. Any user specified function should expect a vector of balance statistics ("p-values") and it should return either a vector of values (in which case "lexical" optimization will be done) or a scalar value (which will be maximized). Some possible alternative functions are `mean` or `median`.

`data.type.integer`
By default, floating-point weights are considered. If this option is set to `TRUE`, search will be done over integer weights. Note that before version 4.1, the default was to use integer weights.

`restrict` A matrix which restricts the possible matches. This matrix has one row for each restriction and three columns. The first two columns contain the two observation numbers which are to be restricted (for example 4 and

20), and the third column is the restriction imposed on the observation-pair. Negative numbers in the third column imply that the two observations cannot be matched under any circumstances, and positive numbers are passed on as the distance between the two observations for the matching algorithm. The most commonly used positive restriction is `0` which implies that the two observations will always be matched.

Exclusion restriction are even more common. For example, if we want to exclude the observation pair 4 and 20 and the pair 6 and 55 from being matched, the restrict matrix would be: `restrict=rbind(c(4,20,-1),c(6,55,-1))`

`cluster` This can either be an object of the 'cluster' class returned by one of the `makeCluster` commands in the snow package or a vector of machine names so that `GenMatch` can setup the cluster automatically. If it is the latter, the vector should look like:
`c("localhost","musil","musil","deckard")`.
This vector would create a cluster with four nodes: one on the localhost another on "deckard" and two on the machine named "musil". Two nodes on a given machine make sense if the machine has two or more chips/cores. `GenMatch` will setup a SOCK cluster by a call to `makeSOCKcluster`. This will require the user to type in her password for each node as the cluster is by default created via `ssh`. One can add on usernames to the machine name if it differs from the current shell: "username@musil". Other cluster types, such as PVM and MPI, which do not require passwords, can be created by directly calling `makeCluster`, and then passing the returned cluster object to `GenMatch`. For an example of how to manually setup up a cluster with a direct call to `makeCluster` see http://sekhon.berkeley.edu/matching/R/cluster_manual.R. For an example of how to get around a firewall by ssh tunneling see: http://sekhon.berkeley.edu/matching/R/cluster_manual_tunnel.R.

`balance` This logical flag controls if load balancing is done across the cluster. Load balancing can result in better cluster utilization; however, increased communication can reduce performance. This option is best used if each individual call to `Match` takes at least several minutes to calculate or if the nodes in the cluster vary significantly in their performance. If cluster==FALSE, this option has no effect.

`...` Other options which are passed on to `genoud`.

Details

Value

`value` The fit values at the solution. By default, this is a vector of p-values sorted from the smallest to the largest. There will generally be twice as many p-values as there are variables in `BalanceMatrix`, unless there are dichotomous variables in this matrix. There is one p-value for each covariate in `BalanceMatrix` which is the result of a paired t-test and another p-value for each non-dichotomous variable in `BalanceMatrix` which is the result of a Kolmogorov-Smirnov test. Recall that these p-values cannot be interpreted as hypothesis tests. They are simply measures of balance.

`par` A vector of the weights given to each variable in `X`.

`Weight.matrix` A matrix whose diagonal corresponds to the weight given to each variable in `X`. This object corresponds to the `Weight.matrix` in the `Match` function.

`matches` A matrix where the first column contains the row numbers of the treated observations in the matched dataset. The second column contains the row numbers of the control observations. And the third column contains the weight that each matched pair is given. These columns correspond respectively to the `index.treated`, `index.control` and `weights` objects which are returned by `Match`.

`ecaliper` The size of the enforced caliper on the scale of the `X` variables. This object has the same length as the number of covariates in `X`.

Author(s)

Jasjeet S. Sekhon, UC Berkeley, `sekhon@berkeley.edu`, http://sekhon.berkeley.edu/.

References

Sekhon, Jasjeet S. 2007. "Multivariate and Propensity Score Matching Software with Automated Balance Optimization." *Journal of Statistical Software*. http://sekhon.berkeley.edu/papers/MatchingJSS.pdf

Sekhon, Jasjeet S. 2006. "Alternative Balance Metrics for Bias Reduction in Matching Methods for Causal Inference." Working Paper. http://sekhon.berkeley.edu/papers/SekhonBalanceMetrics.pdf

Diamond, Alexis and Jasjeet S. Sekhon. 2005. "Genetic Matching for Estimating Causal Effects: A General Multivariate Matching Method for Achieving Balance in Observational Studies." Working Paper. http://sekhon.berkeley.edu/papers/GenMatch.pdf

Sekhon, Jasjeet Singh and Walter R. Mebane, Jr. 1998. "Genetic Optimization Using Derivatives: Theory and Application to Nonlinear Models." *Political Analysis*, 7: 187-210. http://sekhon.berkeley.edu/genoud/genoud.pdf

See Also

Also see `Match`, `summary.Match`, `MatchBalance`, `genoud`, `balanceMV`, `balanceUV`, `qqstats`, `ks.boot`, `GerberGreenImai`, `lalonde`

Examples

```
data(lalonde)
attach(lalonde)

#The covariates we want to match on
X = cbind(age, educ, black, hisp, married, nodegr, u74, u75, re75, re74)

#The covariates we want to obtain balance on
BalanceMat <- cbind(age, educ, black, hisp, married, nodegr, u74, u75, re75, re74,
                    I(re74*re75))

#
#Let's call GenMatch() to find the optimal weight to give each
#covariate in 'X' so as we have achieved balance on the covariates in
#'BalanceMat'. This is only an example so we want GenMatch to be quick
#so the population size has been set to be only 16 via the 'pop.size'
#option. This is *WAY* too small for actual problems.
#For details see http://sekhon.berkeley.edu/papers/MatchingJSS.pdf.
#
genout <- GenMatch(Tr=treat, X=X, BalanceMatrix=BalanceMat, estimand="ATE", M=1,
                   pop.size=16, max.generations=10, wait.generations=1)

#The outcome variable
Y=re78/1000

#
# Now that GenMatch() has found the optimal weights, let's estimate
# our causal effect of interest using those weights
#
mout <- Match(Y=Y, Tr=treat, X=X, estimand="ATE", Weight.matrix=genout)
summary(mout)

#
#Let's determine if balance has actually been obtained on the variables of interest
#
mb <- MatchBalance(treat~age +educ+black+ hisp+ married+ nodegr+ u74+ u75+
                   re75+ re74+ I(re74*re75),
                   match.out=mout, nboots=500, ks=TRUE, mv=FALSE)

# For more examples see: http://sekhon.berkeley.edu/matching/R.
```

D. MatchBalance: Tests for Univariate and Multivariate Balance

`MatchBalance` *Tests for Univariate and Multivariate Balance*

Description

This function provides a variety of balance statistics useful for determining if balance exists in any unmatched dataset and in matched datasets produced by the `Match` function. Matching is performed by the `Match` function, and `MatchBalance` is used to determine if `Match` was successful in achieving balance on the observed covariates.

Usage

```
MatchBalance(formul, data = NULL, match.out = NULL, ks = TRUE, mv = FALSE,
             nboots=500, nmc=nboots,  maxit = 1000,
             weights=NULL, digits=5, paired=TRUE, print.level=1, ...)
```

Arguments

`formul` This formula does *not* estimate any model. The formula is simply an efficient way to use the R modeling language to list the variables we wish to obtain univariate balance statistics for. The dependent variable in the formula is usually the treatment indicator. One should include many functions of the observed covariates. Generally, one should request balance statistics on more higher-order terms and interactions than were used to conduct the matching itself.

`data` A data frame which contains all of the variables in the formula. If a data frame is not provided, the variables are obtained via lexical scoping.

`match.out` The output object from the `Match` function. If this output is included, `MatchBalance` will provide balance statistics for both before and after matching. Otherwise balance statistics will only be reported for the raw unmatched data.

`ks` A logical flag for whether the univariate bootstrap Kolmogorov-Smirnov (KS) test should be calculated. If the ks option is set to true, the univariate KS test is calculated for all non-dichotomous variables. The bootstrap KS test is consistent even for non-continuous variables. See `ks.boot` for more details.

`mv` A logical flag for whether multivariate balance tests (the Kolmogorov-Smirnov and Chi-Square tests) should be calculated. If this flag is `TRUE`, then the formula provided to `MatchBalance` will be used to estimate a logistic regression. And the multivariate tests will be conducted on the

predicted probabilities of treatment for both treated and control based on the formula. The predicted probability densities for both treated and control should be indistinguishable if balance has been achieved. The model defined by this formula is estimated separately for the matched and unmatched datasets.

`maxit` The maximum number of iterations for the glm logistic procedure.

`weights` An optional vector of observation specific weights.

`nboots` The number of bootstrap samples to be run. If zero, no bootstraps are done. Bootstrapping is highly recommended because the bootstrapped Kolmogorov-Smirnov test provides correct coverage even when the distributions being compared are not continuous. At least 500 `nboots` (preferably 1000) are recommended for publication quality p-values.

`nmc` This option is only used if the `mv` flag is `TRUE`. The number of Monte Carlo simulations to be conducted for each multivariate Kolmogorov-Smirnov test calculated. Monte Carlo simulations are highly recommended because the usual Kolmogorov-Smirnov test is not consistent when the densities being compared contain point masses. At least 500 `nmc` (preferably 1000) are recommended for publication quality p-values. Also see the `nboots` option.

`digits` The number of significant digits that should be displayed.

`paired` A flag for whether the paired `t.test` should be used after matching. Regardless of the value of this option, an unpaired `t.test` is done for the unmatched data because it is assumed that the unmatched data were not generated by a paired experiment.

`print.level` The amount of printing to be done. If zero, there is no printing. If one, the results are summarized. If two, details of the computations are printed.

`...` Further arguments passed to `balanceMV`.

Details

This function can be used to determine if there is balance in the pre- and/or post-matching datasets. Difference of means between treatment and control groups are provided as well as a variety of summary statistics for the empirical CDF (eCDF) and empirical-QQ (eQQ) plot between the two groups. The eCDF results are the standardized mean, median and maximum differences in the empirical CDF. The eQQ results are summaries of the raw differences in the empirical-QQ plot.

Two univariate tests are also provided: the t-test and the bootstrap Kolmogorov-Smirnov (KS) test. These tests should not be treated as hypothesis tests in the usual fashion because we wish to maximize balance without limit. The bootstrap KS test is highly recommended (see the `ks` and `nboots` options) because the bootstrap KS is consistent even for non-continuous distributions. Before matching, the two sample t-test is used;

after matching, the paired t-test is used.

Two multivariate tests are provided. The KS and Chi-Square null deviance tests. The KS test is to be preferred over the Chi-Square test because the Chi-Square test is not testing the relevant hypothesis. The null hypothesis for the KS test is equal balance in the estimated probabilities between treated and control. The null hypothesis for the Chi-Square test, however, is all of the parameters being insignificant; a comparison of residual versus null deviance. If the covariates being considered are discrete, this KS test is asymptotically nonparametric as long as the logit model does not produce zero parameter estimates.

`NA`'s are handled by the `na.action` option. But it is highly recommended that `NA`'s not simply be deleted, but one should check to make sure that missingness is balanced.

Value

`BeforeMatching`
: A list containing the before matching univariate balance statistics. That is, a list containing the results of the `balanceUV` function applied to all of the covariates described in `formul`. Note that the univariate test results for all of the variables in `formul` are printed if `verbose > 0`.

`AfterMatching`
: A list containing the after matching univariate balance statistics. That is, a list containing the results of the `balanceUV` function applied to all of the covariates described in `formul`. Note that the univariate test results for all of the variables in `formul` are printed if `verbose > 0`. This object is `NULL`, if no matched dataset was provided.

`BMsmallest.p.value`
: The smallest p.value found across all of the *before* matching balance tests (including t-tests and KS-tests.

`BMsmallestVarName`
: The name of the variable with the `BMsmallest.p.value` (a vector in case of ties).

`BMsmallestVarNumber`
: The number of the variable with the `BMsmallest.p.value` (a vector in case of ties).

`AMsmallest.p.value`
: The smallest p.value found across all of the *after* matching balance tests (including t-tests and KS-tests.

`AMsmallestVarName`
: The name of the variable with the `AMsmallest.p.value` (a vector in case of ties).

`AMsmallestVarNumber`
: The number of the variable with the `AMsmallest.p.value` (a vector in case of ties).

`mv` A return object from a call to `balanceMV`

Author(s)

Jasjeet S. Sekhon, UC Berkeley, `sekhon@berkeley.edu`, http://sekhon.berkeley.edu/.

References

Sekhon, Jasjeet S. 2007. "Multivariate and Propensity Score Matching Software with Automated Balance Optimization." *Journal of Statistical Software*. http://sekhon.berkeley.edu/papers/MatchingJSS.pdf

Sekhon, Jasjeet S. 2006. "Alternative Balance Metrics for Bias Reduction in Matching Methods for Causal Inference." Working Paper. http://sekhon.berkeley.edu/papers/SekhonBalanceMetrics.pdf

Diamond, Alexis and Jasjeet S. Sekhon. 2005. "Genetic Matching for Estimating Causal Effects: A General Multivariate Matching Method for Achieving Balance in Observational Studies." Working Paper. http://sekhon.berkeley.edu/papers/GenMatch.pdf

Abadie, Alberto. 2002. "Bootstrap Tests for Distributional Treatment Effects in Instrumental Variable Models." *Journal of the American Statistical Association*, 97:457 (March) 284-292.

Hall, Peter. 1992. *The Bootstrap and Edgeworth Expansion*. New York: Springer-Verlag.

Wilcox, Rand R. 1997. *Introduction to Robust Estimation*. San Diego, CA: Academic Press.

William J. Conover (1971), *Practical nonparametric statistics*. New York: John Wiley & Sons. Pages 295-301 (one-sample "Kolmogorov" test), 309-314 (two-sample "Smirnov" test).

Shao, Jun and Dongsheng Tu. 1995. *The Jackknife and Bootstrap*. New York: Springer-Verlag.

See Also

Also see `Match`, `GenMatch`, `balanceMV`, `balanceUV`, `qqstats`, `ks.boot`, `GerberGreenImai`, `lalonde`

Examples

```
#
# Replication of Dehejia and Wahba psid3 model
#
# Dehejia, Rajeev and Sadek Wahba. 1999.``Causal Effects in Non-Experimental Studies:
```

```
# Re-Evaluating the Evaluation of Training Programs.''Journal of the American
# Statistical Association 94 (448): 1053-1062.
#
data(lalonde)

#
# Estimate the propensity model
#
glm1  <- glm(treat~age + I(age^2) + educ + I(educ^2) + black +
             hisp + married + nodegr + re74  + I(re74^2) + re75 + I(re75^2) +
             u74 + u75, family=binomial, data=lalonde)

#
#save data objects
#
X  <- glm1$fitted
Y  <- lalonde$re78
Tr  <- lalonde$treat

#
# one-to-one matching with replacement (the "M=1" option).
# Estimating the treatment effect on the treated (the "estimand" option which defaults to 0).
#
rr  <- Match(Y=Y,Tr=Tr,X=X,M=1);

#Let's summarize the output
summary(rr)

#
# Let's check for balance
# 'nboots' and 'nmc' are set to small values in the interest of speed.
# Please increase to at least 500 each for publication quality p-values.
mb  <- MatchBalance(treat~age + I(age^2) + educ + I(educ^2) + black +
                    hisp + married + nodegr + re74  + I(re74^2) + re75 + I(re75^2) +
                    u74 + u75, data=lalonde, match.out=rr, nboots=10, nmc=10)
```

References

Abadie A (2002). "Bootstrap Tests for Distributional Treatment Effect in Instrumental Variable Models." *Journal of the American Statistical Association*, **97**(457), 284–292.

Abadie A, Imbens G (2006). "Large Sample Properties of Matching Estimators for Average Treatment Effects." *Econometrica*, **74**, 235–267.

Barnard J, Frangakis CE, Hill JL, Rubin DB (2003). "Principal Stratification Approach to Broken Randomized Experiments: A Case Study of School Choice Vouchers in New York City." *Journal of the American Statistical Association*, **98**(462), 299–323.

Bonney J, Canes-Wrone B, Minozzi W (2007). "Issue Accountability and the Mass Public: The Electoral Consequences of Legislative Voting on Crime Policy." Working Paper.

Bowers J, Hansen B (2005). "Attributing Effects to A Cluster Randomized Get-Out-The-Vote Campaign." Technical Report #448, Statistics Department, University of Michigan. http://www-personal.umich.edu/~jwbowers/PAPERS/bowershansen2006-10TechReport.pdf.

Brady H, Hui I (2006). "Is it Worth Going the Extra Mile to Improve Causal Inference?"." Paper presented at the 23rd Annual Summer Meeting of the Society of Political Methodology.

Christakis NA, Iwashyna TI (2003). "The Health Impact of Health Care on Families: A matched cohort study of hospice use by decedents and mortality outcomes in surviving, widowed spouses." *Social Science & Medicine*, **57**(3), 465–475.

Cochran WG, Rubin DB (1973). "Controlling Bias in Observational Studies: A Review." *Sankhya,* Ser. A, **35**, 417–446.

Dawid AP (1979). "Conditional Independence in Statistical Theory." *Journal of the Royal Statistical Society, Series B*, **41**(1), 1–31.

Dehejia R (2005). "Practical Propensity Score Matching: A Reply to Smith and Todd." *Journal of Econometrics*, **125**(1–2), 355–364.

Dehejia R, Wahba S (1997). "Causal Effects in Non-Experimental Studies: Re-Evaluating the Evaluation of Training Programs." Rejeev Dehejia, *Econometric Methods for Program Evaluation*. Ph.D. Dissertation, Harvard University, Chapter 1.

Dehejia R, Wahba S (1999). "Causal Effects in Non-Experimental Studies: Re-Evaluating the Evaluation of Training Programs." *Journal of the American Statistical Association*, **94**(448), 1053–1062.

Dehejia RH, Wahba S (2002). "Propensity Score Matching Methods for Nonexperimental Causal Studies." *Review of Economics and Statistics*, **84**(1), 151–161.

Diamond A, Sekhon JS (2005). "Genetic Matching for Estimating Causal Effects: A General Multivariate Matching Method for Achieving Balance in Observational Studies." Working Paper.

Diprete TA, Engelhardt H (2004). "Estimating Causal Effects With Matching Methods in the Presence and Absence of Bias Cancellation." *Sociological Methods & Research*, **32**(4), 501–528.

Fechner GT (1966 [1860]). *Elements of psychophysics, Vol 1.* Rinehart & Winston, New York. Translated by Helmut E. Adler and edited by D.H. Howes and E.G. Boring.

Galiani S, Gertler P, Schargrodsky E (2005). "Water for Life: The Impact of the Privatization of Water Services on Child Mortality." *Journal of Political Economy*, **113**(1), 83–120.

Gilligan MJ, Sergenti EJ (2006). "Evaluating UN Peacekeeping with Matching to Improve Causal Inference." http://sekhon.berkeley.edu/causalinf/papers/gilligan_sergenti_06.pdf.

Gordon S, Huber G (2007). "The Effect of Electoral Competitiveness on Incumbent Behavior." *Quarterly Journal of Political Science*, **2**(2), 107–138.

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

Hansen BB, Klopfer SO (2006). "Optimal full matching and related designs via network flows." *Journal of Computational and Graphical Statistics*, **15**, 609–627.

Heckman JJ, Ichimura H, Smith J, Todd P (1998). "Characterizing Selection Bias Using Experimental Data." *Econometrica*, **66**(5), 1017–1098.

Herron MC, Wand J (forthcoming). "Assessing Partisan Bias in Voting Technology: The Case of the 2004 New Hampshire Recount." *Electoral Studies*.

Holland PW (1986). "Statistics and Causal Inference." *Journal of the American Statistical Association*, **81**(396), 945–960.

Horvitz DG, Thompson DJ (1952). "A Generalization of Sampling without Replacement from a Finite Universe." *Journal of the American Statistical Association*, **47**, 663–685.

Imai K (2005). "Do Get-Out-The-Vote Calls Reduce Turnout? The Importance of Statistical Methods for Field Experiments." *American Political Science Review*, **99**(2), 283–300.

LaLonde R (1986). "Evaluating the Econometric Evaluations of Training Programs with Experimental Data." *American Economic Review*, **76**, 604–20.

Lenz GS, Ladd JM (2006). "Exploiting a Rare Shift in Communication Flows: Media Effects in the 1997 British Election." http://sekhon.berkeley.edu/causalinf/papers/LaddLenzBritish.pdf.

Mebane WRJ, Sekhon JS (1998). "GENetic Optimization Using Derivatives (GENOUD)." Software Package. http://sekhon.berkeley.edu/rgenoud/.

Morgan SL, Harding DJ (2006). "Matching Estimators of Causal Effects: Prospects and Pitfalls in Theory and Practice." *Sociological Methods & Research*, **35**(1), 3–60.

Nix AE, Vose MD (1992). "Modeling Genetic Algorithms with Markov Chains." *Annals of Mathematics and Artificial Intelligence*, **5**, 79–88.

Park JH (2006). "Causal Effect of Information on Voting Behavior from a Natural Experiment: An Analysis of Candidate Blacklisting Campaign in 2000 South Korean National Assembly Election." Working paper.

Raessler S, Rubin DB (2005). "Complications when using nonrandomized job training data to draw causal inferences." *Proceedings of the International Statistical Institute*.

Rosenbaum PR (1989). "Optimal Matching for Observational Studies." *Journal of the American Statistical Association*, **84**(408), 1024–1032.

Rosenbaum PR (1991). "A Characterization of Optimal Designs for Observational Studies." *Journal of the Royal Statistical Society, Series B*, **53**(3), 597–610.

Rosenbaum PR (2002). *Observational Studies*. Springer-Verlag, New York, 2nd edition.

Rosenbaum PR, Rubin DB (1983). "The Central Role of the Propensity Score in Observational Studies for Causal Effects." *Biometrika*, **70**(1), 41–55.

Rosenbaum PR, Rubin DB (1985). "Constructing a Control Group Using Multivariate Matched Sampling Methods That Incorporate the Propensity Score." *The American Statistician*, **39**(1), 33–38.

Rubin DB (1974). "Estimating Causal Effects of Treatments in Randomized and Nonrandomized Studies." *Journal of Educational Psychology*, **66**, 688–701.

Rubin DB (1976a). "Multivariate Matching Methods That are Equal Percent Bias Reducing, I: Some Examples." *Biometrics*, **32**(1), 109–120.

Rubin DB (1976b). "Multivariate Matching Methods That are Equal Percent Bias Reducing, II: Maximums on Bias Reduction for Fixed Sample Sizes." *Biometrics*, **32**(1), 121–132.

Rubin DB (1977). "Assignment to a Treatment Group on the Basis of a Covariate." *Journal of Educational Statistics*, **2**, 1–26.

Rubin DB (1978). "Bayesian Inference for Causal Effects: The Role of Randomization." *Annals of Statistics*, **6**(1), 34–58.

Rubin DB (1979). "Using Multivariate Sampling and Regression Adjustment to Control Bias in Observational Studies." *Journal of the American Statistical Association*, **74**, 318–328.

Rubin DB (1980). "Bias Reduction Using Mahalanobis-Metric Matching." *Biometrics*, **36**(2), 293–298.

Rubin DB (1990). "Comment: Neyman (1923) and Causal Inference in Experiments and Observational Studies." *Statistical Science*, **5**(4), 472–480.

Rubin DB (1997). "Estimating Causal Effects from Large Data Sets Using Propensity Scores." *Annals of Internal Medicine*, **127**(8S), 757–763.

Rubin DB (2001). "Using Propensity Scores to Help Design Observational Studies: Application to the Tobacco Litigation." *Health Services & Outcomes Research Methodology*, **2**(1), 169–188.

Rubin DB (2006). *Matched Sampling for Causal Effects*. Cambridge University Press, New York.

Rubin DB, Stuart EA (2006). "Affinely Invariant Matching Methods with Discriminant Mixtures of Proportional Ellipsoidally Symmetric Distributions." *Annals of Statistics*. In Press.

Rubin DB, Thomas N (1992). "Affinely Invariant Matching Methods with Ellipsoidal Distributions." *Annals of Statistics*, **20**(2), 1079–1093.

Sekhon JS (2004). "The Varying Role of Voter Information Across Democratic Societies." Working Paper, URL http://sekhon.berkeley.edu/papers/SekhonInformation.pdf.

Sekhon JS (2006a). "Alternative Balance Metrics for Bias Reduction in Matching Methods for Causal Inference." Working Paper, URL http://sekhon.berkeley.edu/papers/SekhonBalanceMetrics.pdf.

Sekhon JS (2006b). "The Art of Benchmarking: Evaluating the Performance of R on Linux and OS X." *The Political Methodologist*, **14**(1).

Sekhon JS, Mebane Jr WR (1998). "Genetic Optimization Using Derivatives: Theory and Application to Nonlinear Models." *Political Analysis*, **7**, 189–203.

Smith HL (1997). "Matching with Multiple Controls to Estimate Treatment Effects in Observational Studies." *Sociological Methodology*, **27**, 305–353.

Smith J, Todd P (2005a). "Does Matching Overcome LaLonde's Critique of Nonexperimental Estimators?" *Journal of Econometrics*, **125**(1–2), 305–353.

Smith J, Todd P (2005b). "Rejoinder." *Journal of Econometrics*, **125**(1–2), 365–375.

Smith JA, Todd PE (2001). "Reconciling Conflicting Evidence on the Performance of Propensity Score Matching Methods." *AEA Papers and Proceedings*, **91**(2), 112–118.

Splawa-Neyman J (1923). "On the Application of Probability Theory to Agricultural Experiments. Essay on Principles. Section 9." *Statistical Science*, **5**(4), 465–472. Trans. Dorota M. Dabrowska and Terence P. Speed.

Vose MD (1993). "Modeling Simple Genetic Algorithms." In LD Whitley (ed.), "Foundations of Genetic Algorithms 2," Morgan Kaufmann, San Mateo, CA.

Winship C, Morgan S (1999). "The estimation of causal effects from observational data." *Annual Review of Sociology*, **25**, 659–707.

Affiliation:

Jasjeet S. Sekhon
Associate Professor of Political Science
Survey Research Center
2538 Channing Way
UC Berkeley
Berkeley, CA 94720-5100
E-mail: sekhon@berkeley.edu
URL: http://sekhon.berkeley.edu

This work is licensed under a Creative Commons Attribution-No Derivative Works 3.0 License

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume VV, Issue II *Submitted:* yyyy-mm-dd
MMMMMM YYYY *Accepted:* yyyy-mm-dd