Development of a Python Package for Matching Observational Data

Jack Potrykus

April 29, 2022

Abstract

Observational studies are widely used throughout econometrics, psychology, and medical research. Matching is a field in causal statistics concerned with algorithms to minimize the effect of selection bias in the observational data on analyses of treatment effects. In particular, in binary treatment/control studies, these algorithms work by matching each treatment observation to one or more "nearby" control observations. This paper breaks the matching procedure down into two key components: how distance is measured, and how matches are assigned. In doing so, it explores several distance metrics, in particular the propensity score (Rosenbaum and Rubin 1983), the prognostic score (Ben B. Hansen 2008), and exact matching (Iacus, King, and Porro 2012) and its machine learning extensions (Liu et al. 2019; Wang et al. 2021); it then explores several matching algorithms that can be used to produce the matched subset once the distance is calculated, in particular the Hungarian algorithm (Munkres 1957) and greedy algorithms (D. Ho et al. 2011). I then showcase the functionality of `matching`, an open-source Python package I have developed for matching observational data which takes a graph-centric approach, something which no other package offers. I finally explore some practical considerations of parameter tuning when matching via experiments on simulated data.

Contents

1 Introduction

Observational studies are of ever-increasing importance as the numerous amount of data collected each year continues to grow. However, of observational studies are at a major disadvantage compared to blocking-based study design or randomized control trials (RCTs), because the researchers do not have control over the distribution of confounding variables, known or unknown, in the data itself. As such, researchers must take preliminary steps to alter the data in some way in order to minimize the effects of selection bias, which results from heterogeneous distributions present in the observational data.

Within the field of causal inference, the term *matching* is often associated with observational studies with a binary treatment indicator. Using a similar framework to that of Iacus, King, and Porro (2011), consider $X_T \in \mathbb{R}^{n \times k}$ and $X_C \in \mathbb{R}^{m \times k}$, matrices of k features for n treatment-group observations and m control-group observations, $n \leq m$. We seek to produce a matching $\mathcal{M}$ such that each observation in X_T is paired with one or more nearby observations from X_C, according to a distance metric and a matching algorithm.

The choice of distance measure and choice of matching algorithm are orthogonal, and this paper (and the Python package) will discuss them as such. Indeed, measuring distance, as well as *balance diagnostics* between treatment and control distributions, are problems most associated with the field of statistics; in contrast, producing the optimally matched subset is a *minimum cost flow* problem in the field of computer science (Rosenbaum 1989).

I then introduce a Python package I have developed, `matching`, which provides an idiomatic and flexible tool for matching observational data. It achieves this by allowing distance measures and matching algorithms to be independently and iteratively applied, so that even incredibly bespoke matching procedures may be implemented succinctly in code. It is also the first Python library for matching observational data which is "graph-centric": the key data structure used is a bipartite graph, whose nodes represent observation IDs, and edge weights the distance measure between them. This enables further exploration into new matching procedures and balance diagnostics using graph metrics and algorithms from computer science.

2 Literature Review

2.1 Measuring Distance

In the context of this paper, distance measures are functions $d : \mathbb{R}^k \times \mathbb{R}^k \mapsto \mathbb{R}^+$ which produce a non-negative distance, or "cost of matching", between any vectors $\mathbf{x}_T, \mathbf{x}_C$ from the rows of $X_T \in \mathbb{R}^{n \times k}$ and $X_C \in \mathbb{R}^{m \times k}$, to judge the match quality. Distance measures are often combined with some preprocessing function $f(X)$, usually in the form of a dimension reduction or discretization of the data. Again using a framework akin to that of Iacus, King, and Porro (2011), one may succinctly describe a distance measurement as

$$\mathcal{D}_d(f(X_T), f(X_C)), \tag{1}$$

where $\mathcal{D}_d : \mathbb{R}^{n \times k} \times \mathbb{R}^{m \times k} \mapsto \mathbb{R}^{n \times m}$ is a "vectorized" version of d, producing a matrix D whose (i, j)th element is the distance between observation i from X_T and observation j from X_C.

In the simplest case, d is some norm (perhaps Euclidean) calculated on the difference between feature vectors, and f is the identity. With this simple framework in mind, we are now ready to explore extensions of the simple case, and examine their proposed benefits.

2.1.1 Balancing Scores

Balancing scores [1] (Rosenbaum and Rubin 1983) are a versatile class of functions $b(X)$ satisfying the property

$$X \perp \mathbf{z} | b(X), \tag{2}$$

where the matrix $X \in \mathbb{R}^{(n+m)\times k} = X_T \cup X_C$, and the vector $\mathbf{z} \in \mathbb{R}^{n+m}$ contains binary treatment group assignments. In words, given the balancing score $b(X)$, the distribution of the feature vector X is independent of treatment assignment $\mathbf{z}$. These methods each set d equal to the L^1 norm, or absolute, distance by convention. In the notation of (1), then, balancing scores compose the class of distance measures of the form

$$\mathcal{D}_{L^1}(b(X)_T, b(X)_C). \tag{3}$$

TODO DISCUSS STRONG IGNORABILITY, PROPERTIES, WHEN IS ATE ESTIMATE UNBIASED + TODO MAKE CLEAR IT IS REPLICATING A RANDOMIZED TRIAL

The Propensity Score. In the same paper, Rosenbaum and Rubin (1983) propose the balancing score $b(X) = \mathbb{E}[\mathbf{z}|X]$, also known as the propensity score. This is simply a predicted probability (or logit) that any given feature vector $\mathbf{x} \in X$ belongs to the treatment group. This is the most prevalent balancing score in the literature, and indeed the most prevalent distance measurement overall. It is almost always estimated by logistic or probit regression (Garrido et al. 2014), but other classifiers are possible. By preprocessing X to a single dimension of scores $b(X)$, we not only reduce computation time necessary for $\mathcal{D}$, but also implicitly apply a weighting to the feature matrix X: the features most predictive of treatment assignment (i.e., most heterogeneous between the two groups) will be the most closely matched. Dehejia and Wahba 1999 evaluated the performance of propensity score methods, and found that, given the assumption that treatment depends only on pre-intervention observable features, propensity score methods can serve as a useful diagnostic, particularly for examining the degree of "overlap" in feature distributions between the two groups.

Extensions of the Propensity Score. Acknowledging that propensity score methods have thus far been limited to binary, multinomial, or ordinal treatment assignments in the literature (as they are considered in this paper), Imai and Dyk (2004) explored extensions of the propensity score into the realm of quantitative treatments. In their paper, they consider the treatment vector *packyear*, the number of packs smoked by a smoker each year, and subclass the resulting scores into a varying number of bins. They found that conditioning on the subclass helped to reduce bias by between 16% and 95%.

The Prognostic Score. Ben B. Hansen (2008) discussed the *prognostic score*, a balancing metric on outcomes. The only difference between the construction in (2) is that we exchange $\mathbf{z}$ for $\mathbf{y}$, a vector of outcomes.

$$X \perp \mathbf{y} | b(X) \iff b \text{ is a prognostic balancing score} \tag{4}$$

The prognostic score is constructed nearly in exactly the same way as the propensity score; however, in order to preserve (4), the model (linear/logistic regression, random forest, etc.) is trained on the *control data alone*. This model is then used to predict scores over the *whole dataset*.

The idea of using outcomes, either raw or "preprocessed", does not see unanimous support; Garrido et al. (2014) argues against incorporating any information about outcome into the matching process.

[1] Balancing scores come in a variety of forms: as Rosenbaum and Rubin (1983) note, $b(X) = X$ is the simplest balancing score. However, this paper uses the term "balancing score" specifically with regards to 1-dimensional reductions of the data. That is to say, that we will consider balancing scores as a preprocessing function $b : \mathbb{R}^{(n+m)\times k} \mapsto \mathbb{R}^{n+m}$.

Miettinen (1976) was a seminal paper in matching that used outcome stratification; this method has since been found to be suboptimal, and has been effectively replaced by prognostic scores and other methods (Ben B Hansen 2006) (Miettinen's "multivariate confounder score" used the *full* dataset, not just the control, when fitting). Ben B Hansen (2006) argues for its inclusion via a "conditionality principle", which suggests that if some statistic (in this case, $b(X)$) is known to be uninformative about the parameter being estimated (i.e. treatment effects), it should be included in inferential models.

Stuart, Lee, and Leacy (2013) found prognostic scores to be highly correlated with bias in estimates of treatment effects, and thus a useful diagnostic tool, even when the model was misspecified. Thus, prognostic scores can serve as a useful "proxy" for the reduction in bias of the estimates. Nonetheless, they also caution that their use in matching depends on the researcher's appetite to reduce the separation of data identification and analysis of outcomes in their study.

Joint Use of Balancing Scores. Leacy and Stuart (2014) further explores of calculating *both* propensity and prognostic scores, stratifying each into a $k \times k$ grid, and considering observations within each of the k^2 grid squares as a *matched strata*. However, they found that matching on prognostic scores alone performed the best, while matching on propensity scores and Mahalanobis distance (see §2.1.3) also performed admirably. The $k \times k$ strata performed better than stratification of either score alone, but could not beat matching on either un-discretized score alone. They also found that the percent bias of estimates created using this matching method increased with the non-linearity in the outcomes model.

Calipers. Propensity and prognostic scoring methods often add one more hyperparameter: a *caliper*, c. The caliper is used to calculate *caliper width*, equal to the product of c and the standard deviation of the scores (D. Ho et al. 2011). The caliper serves to establish a radius of acceptable matches around each observation's propensity score as a form of quality control: if the observations are too dissimilar (their distance is greater than the caliper width), then they may not match. This comes with a cost: if the caliper is sufficiently small, observations with outlying feature vectors $\mathbf{x}$ may not have any suitable matches, and the matched subset remaining for treatment effect analyses may be substantially smaller.

Austin (2011) explored the question of choosing an optimal caliper width under a variety of regimes. When all features were iid Gaussian random variables, calipers $c \in [0.05, 0.15]$ maximized the reduction in bias, with each reduction on the order of 98.9% or greater. Austin then introduced correlation when generating the feature matrix, as well as including a varying number of independent Bernoulli(0.5). Under the correlated regime, the optimal caliper varied between 0.05 and 0.30, depending on the true risk reduction given the generated features. However, he also notes that the percent bias reduction decreased as the number of Bernoulli features increased, and that when *all* features were Bernoulli-distributed, the choice of caliper had no effect on bias reduction.

Criticism. The most notable criticism of balancing score based distance measurement comes from King and Nielsen (2019). King and Nielsen's argument is twofold: first, from an experimental design perspective, they argue that *fully blocked* study design, which other distance measures attempt to emulate (in particular: Mahalanobis distance, Coarsened Exact Matching), dominates *complete randomization study design*, which balancing-score based methods attempt to emulate. Whereas each study design balances unobserved confounding features on average, fully blocked methods assure balance of observed confounding features, whereas complete randomization only promises balance *on average*. They thus regard balancing score methods as having a "lower standard". Second, they observe the "paradox of Propensity Score Matching": consider the case where $X \perp \mathbf{z}$, and so all predicted scores are the same constant: $\hat{\mathbf{z}}$. Then, all potential matches are indecipherable, and the individual matchings may be

complete nonsensical. The authors nonetheless concede that propensity scores (and other balancing scores, by extension) do indeed have nice theoretical properties, and may serve as a useful diagnostic; they contend only that they should not be used for matching.

TODO VERIFY

2.1.2 Exact Matching and Almost Exact Matching

Among other methods which seek to replicate fully blocked study design, King and Nielsen (2019) argue for "coarsened exact matching", or CEM. This requires the researcher to first preprocess their data into discrete categories: for example, a vector "age" might be grouped into buckets. They then suggest exact matching on the now-discretized features into *strata*, and applying observational weights to the data according to the formula

$$w_i = \begin{cases} 1, & i \in \text{treatment group} \\ \frac{m_C}{m_T}\frac{m_T^s}{m_C^s}, & i \in \text{control group} \end{cases}, \tag{5}$$

where w_i denotes the observation weight for observation i, m_T and m_C are the total number of matched treatment and control observations, and m_T^s and m_C^s are the total number of matched treatment and control observations *within observation i's stratum, s* (Iacus, King, and Porro 2012).

Iacus, King, and Porro (2011) introduces a formal definition for a class of distance measures which they call "monotonic imbalance bounding", or MIB. These methods are designed to require no assumptions about the distribution of the data, and to emphasize minimizing *in-sample* imbalance, as opposed to *sample* balance. These methods take a vector of tuning parameters $\boldsymbol{\pi}$, such as a vector of calipers. These methods afford the benefit that the number of matches is a function of the tuning parameters alone, and given monotonicity of the maximum distance function $\gamma_{D_d}(\boldsymbol{\pi})$, can be each adjusted independently other others: if γ is vector-valued, returning a vector in $\mathbb{R}^k$, the distance between each feature vector can be independently compared in a sequence of inequalities. CEM is an example of an MIB distance measure, but this class is clearly much more flexible, allowing for precise adjustments.

Almost Exact Matching. Some of the most modern methods are DAME (Liu et al. 2019) and FLAME Wang et al. 2021, which incorporate machine-learning generated weight vectors indicating the relative importance of features to extend exact matching into the field of *almost exact matching*. These two algorithms are unified by their tendency to sometimes "drop" uninformative features from the matching process, in order to ensure that the most important features are matched on, and potentially increase the number of potential matches.

TODO FINISH

2.1.3 L^p Norms and Mahalanobis Distance

Norms of the difference between feature vectors (i.e. $\|\mathbf{x}_T - \mathbf{x}_C\|_p$) themselves are usually used in the form of Mahalanobis distance, which first rescales all features to have mean zero and unit variance, and then calculates distances using the L^2 norm. This ensures that all features have equal weight in the matching, which may or may not be desirable; in either case, it almost certainly better than the no-rescaling case, where a feature's variance would determine its relative importance. As King and Nielsen (2019) notes, norm-based distance metrics attempt to replicate fully-blocked design, and thus offer some theoretical advantages. However, many are attracted to the ability of balancing score methods, as well as modern methods such as DAME and FLAME, to discern which features are *most important* to match on.

Nonetheless, Leacy and Stuart (2014) finds Mahalanobis distance to perform well when the number of features was small, each of which was approximately normally distributed. Additionally, many modern matching software packages allow for initially establishing a "perimeter" of viable matches using a caliper and a propensity score, and subsequently matching on the Mahalanobis distance between the feature vectors (D. Ho et al. 2011).

2.2 Matching Algorithms

Once the distance measure has been agreed upon, the researcher must then decide *how* to match. A computer scientist could describe this problem as finding the minimum weight full matching of a *bipartite graph*, whose nodes represent observation IDs, and whose edge weights represent the distance between them. Here, the term "bipartite" means that all edges in the graph map $T \leftrightarrow C$; within each disjoint subset, no two nodes share an edge. This paper will explore the optimal solution to this problem in both the $1:1$ and $1:k$ matching scenarios, as well as fast approximations.

2.2.1 Optimal Matching

Rosenbaum (1989) argues for optimal matching according to the Hungarian algorithm, also known as the Kuhn-Munkres or simply Munkres algorithm (Munkres 1957). We will denote a matching a set $\mathcal{M}$ of tuples (i, j), with i representing a treatment-group observation ID, and j a control-group observation ID. The Hungarian algorithm solves the *linear assignment problem*,which in this setting may be expressed as

$$\min_{\mathcal{M}} \sum_{(i,j)\in\mathcal{M}} d(\mathbf{x}_{T_i}, \mathbf{x}_{C_j}), \text{ such that } \forall (i,j),(k,l) \in \mathcal{M}, i = k \iff j = l; \tag{6}$$

the "such that" condition simply specifies that the matching is one-to-one. As such, optimal matching via the Hungarian algorithm minimizes the total sum of distances between matches in the matched subset.

Extensions. The Hungarian algorithm can be extended to the $1:k$ matching case via b-matching while retaining optimality. This process involves adding k "copy" nodes for each node representing a T datapoint, each with identical edges as the original node, and then perform the optimal matching on the resulting graph. However, as Khan et al. (2016) notes, this can lead to calculations which quickly become time consuming as the total number of data points increases. Khan et al. (2016) goes on further to explore efficient approximation algorithms of the b-matching, such as the b-SUITOR algorithm.

2.2.2 Greedy Matching

Greedy algorithms have the benefit of running much quicker than optimal matching algorithms, and are often the default in other observational data matching software, such as `MatchIt` (D. Ho et al. 2011). These methods are equivalent to sorting the edges of the graph in ascending order, and naively popping off the top of the list the next match. While these methods benefit from speed, they guarantee nothing with regards to optimality. This can be particularly problematic in the $1:k$ scenario (Rosenbaum 1989), where the order in matches are assigned can have matches implications; compare this to optimal matching, which is a simultaneous match.

2.3 Balance Assessment

Researchers may wish to assess the balance of their matching, particularly when attempting to heuristically optimize a hyperparameter, such as the caliper. Common numerical methods include standardized mean-differences, variance ratios, and empirical CDF (eCDF) comparison between the two groups (Greifer 2022).

Most researchers do not check for convergence of moments beyond the second, and as Basu, Polsky, and Manning (2008) report, these moments exhibit much slower convergence, the effects of which can be particularly detrimental when outcome depends on some non-linear function of the joint distribution of X. For this reason, Zhu, Savage, and Ghosh (2018) recommend using density-based metrics, such as total variation distance or Kolmogorov-Smirnov distance. It is also important to note that balance of the ith moment does not imply balance of the jth moment across the two groups for $j \geq i$ (Garrido et al. 2014).

There are also many graphical diagnostics for assessing balance. QQ-plots, eCDFs, and density estimates for for the unmatched data and the matched subset, perhaps overlaid, effectively visually convey the balance improvement from matching. One may also plot the improvement in standardized mean differences between the two groups, referred to as a Love plot (Greifer 2022). One could also plot the bipartite graph to understand where clusters are appearing, or how large groups of "similar" observations are, which observations are most "dissimilar" from the rest.

3 `matching`: a Python Package for Matching Observational Data

The most popular implementation of bipartite matching for observational studies is the `MatchIt` package for R (D. Ho et al. 2011). The GitHub repository[2] boasts an impressive 37000 downloads per month, and with good reason: the authors of the package are also authors of many of the most popular papers in the field of matching (e.g. D. E. Ho et al. (2007), Imai and Dyk (2004), King and Nielsen (2019), and Stuart (2010)).

`MatchIt` is largely used through a single function: `Matchit::matchit`, which minimally accepts a `formula` (like those used in `lm`), `data`, and a `method`. The function also allows for iterative matching procedures, complex filters on the suitability of potential matches, and a selection of bipartite matching algorithms. The output of the function is furthermore easily inspected via the ubiquitous `summary` function, or plotted via `plot`.

However, R is not as popular "in-industry" as it once was. Whereas academics have largely favored R as their programming language of choice, Python has become the *lingua franca* of data scientists. The TIOBE index[3] currently lists Python as the most popular programming language in the world, with nearly ten times the market share of R. It is then surprising that there is no package for matching observational data that is as flexible and feature-complete as `MatchIt`. This is what lead me to developing `matching`, a Python library for matching observational data.

3.1 Design Goals

After exploring `MatchIt`'s capabilities, as well as the Python landscape of matching packages for causal inference, I set out to create a package that is:

- inspectable. The user should be able to numerically and graphically inspect each step of the matching procedure.
- flexible. The package should provide "plug-and-play" tools to customize and conduct an arbitrarily complex matching procedure, without the user needing to implement new classes.
- familiar. The package should mimic idioms from popular data science packages[4], and all outputs should be easily coercible to a `pandas.DataFrame` or a `numpy.ndarray` for further analysis.

[2] https://github.com/kosukeimai/MatchIt
[3] https://www.tiobe.com/tiobe-index/
[4] e.g. `numpy`, `pandas`, `sklearn`, `networkx`

- "depth over breadth". The package should only implement matching procedures, leaving model-building and effect estimation to the user. Narrowing in on the matching procedure alone also helps with the first three goals.

In particular, I focused on separating responsibility between how the distance between feature vectors is calculated, and how the matches are then assigned. Being able to easily adjust distance measure and matching procedure independently is especially valuable when assessing the matching quality via diagnostic tools when tuning a hyperparameter, e.g. the maximum allowable distance, or the number of matches k to find for each treatment observation.

3.2 Architecture

The package is organized into the following modules.

- `matching.balance`: implementations of common balance measures, such as standardized mean difference, variance ratio, and eCDFs.
- `matching.dataset`: implementation of the `MatchingDataset` class, which parses user data into an easily manageable format.
- `matching.distance`: the `_Distance` base class and concrete implementations of distance measures, such as `Norm`.
- `matching.graph`: the `MatchingGraph` class implementation; this is the main class that the user interacts with.
- `matching.graph_utils`: functional graph utilities.
- `matching.preprocessing`: functional preprocessing utilities, such as propensity scoring, prognostic scoring, and auto-coarsening.

Most users will only need to use `matching.graph` and `matching.distance`, perhaps in addition to some light preprocessing with `matching.preprocessing` as necessary. These two submodules represent the algorithmic matching, and distance calculations separately. This separation of responsibility into two separate class hierarchies allows for high flexibility in the matching procedure, as each component can be changed independent of the other.

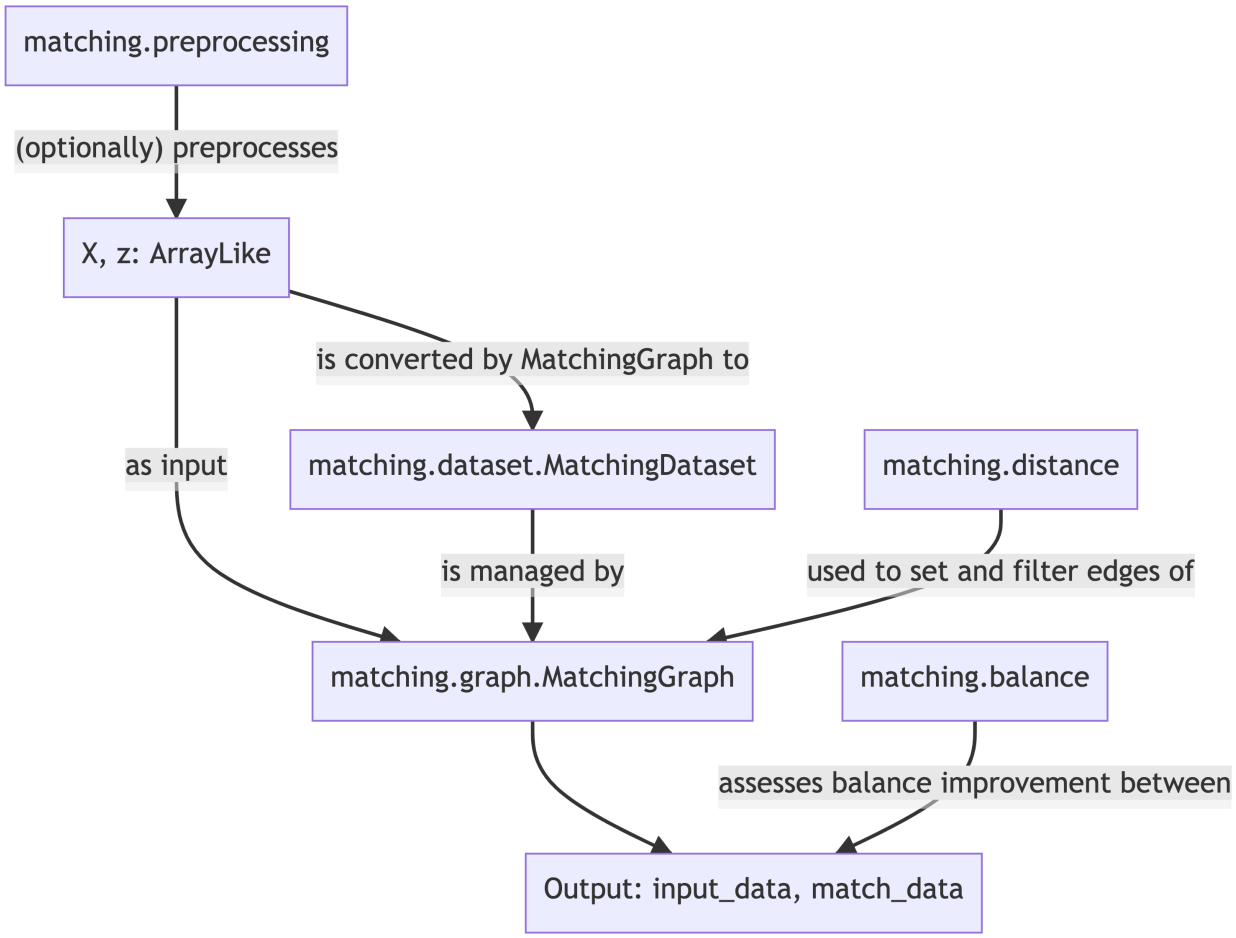

Figure 1: The relationships between the different modules of `matching`

.

3.3 Usage

Typical usage of the `Matching` package is outlined in Figure 3. A written explanation of each step follows.

Initialization. The `MatchingGraph` is minimally initialized with an array-like (e.g., a `pandas.DataFrame`, or a `numpy.ndarray`) of features `X` and an array-like of binary treatment assignments `z`. The user may first wish to preprocess their data (i.e. calculating propensity scores) using `matching.preprocessing`, but this is not necessary.

```
from matching.graph import MatchingGraph
from matching.preprocessing import propensity_score

# Possibly some preprocessing... (not shown)
# X is an array of propensity scores, z is an array of booleans
mg = MatchingGraph(X, z)
```

Setting Edges. The user should then make a call to the `set_edges` method to calculate the distance between each observation the two groups, and set the graph nodes and edge weights. Here, the user

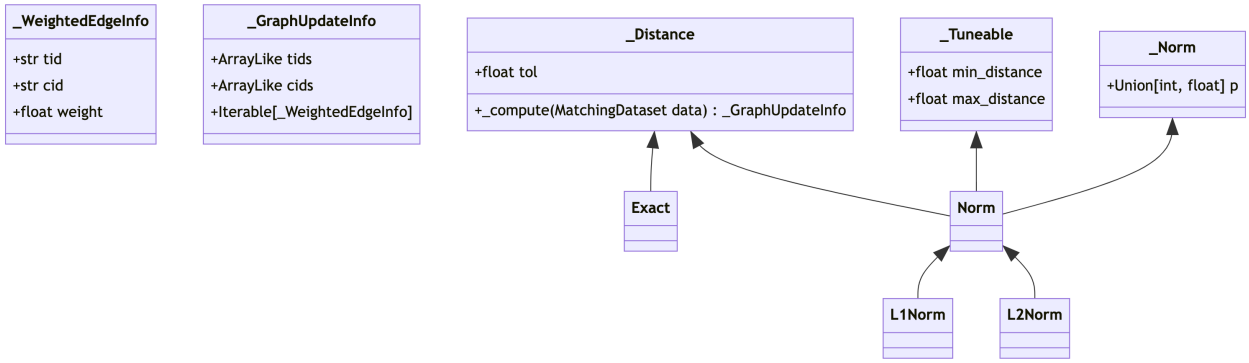

Figure 2: UML Diagram for the `matching.distance` submodule

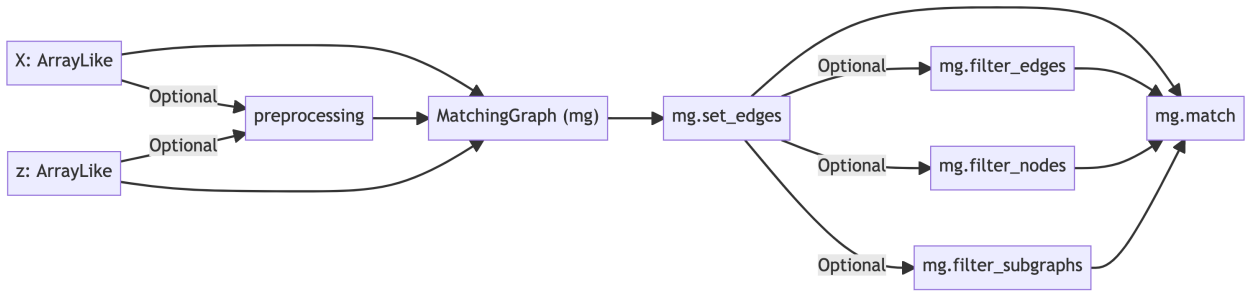

Figure 3: Flow diagram describing typical usage of the `matching` package

supplies a `_Distance` measure from `matching.distance`. The user may supply a `min_distance` and/or a `max_distance` for an edge to be allowed; in graph terms, this can be considered the edges' *capacity*.

```
import numpy as np
from matching.distance import L1Norm

# Calculating the caliper width. Recall that X is an array of scores
caliper = 0.05
caliper_width = caliper * np.std(X)
mg.set_edges(distance=L1Norm(max_distance=caliper_width))
```

Iterative Filtering. It is possible to implement iterative distance measures as well. Consider `X` as a dataframe with columns `"score"`, containing propensity scores, and `"is_nice"`, a boolean series. We would like to match on propensity scores within `caliper_width`, but also cannot assign any nice people to any naughty people. We can make use of the `filter_edges` method and the `include` keyword argument to accomplish this.

```
from matching.distance import Exact, L1Norm
from matching.graph import MatchingGraph
from matching.preprocessing import propensity_score

# Take caliper_width as given
# X has columns "score" and "is_nice", z is treatment assignments
mg = MatchingGraph(X, z)
# NOTE: there is also a keyword argument "exclude"
mg.set_edges(distance=L1Norm(max_distance=caliper_width), include=["score"])
mg.filter_edges(distance=Exact(), include=["is_nice"])
```

The `filter_edges` command will drop any potential matches who do not match exactly on `"is_nice"` by simply deleting the edge between them. This extends beyond exact-match filtering. One could filter using `L2Norm`, for instance, to drop potential matches whose L^2 norm distance was above some maximum threshold.

There are also methods to `filter_nodes` by label or order, as well as to `filter_subgraphs`, which filters the disjoint subgraphs of the graph by a variety of metrics[5]; they will not be discussed here, but readers are referred to `matching` documentation hosted on the GitHub[6].

Matching. Once the user has set the edges of the graph and completed their filtering, they may wish to conduct a $1 : k$ matching between treatment and control, for some integer $k \geq 1$[7]. The user does this via a call to the `match` method. The user can control k with `n_match`, the maximum number of matches to find for each observation. Some treatment observations will not be able to be matched with this many control observations; they can be automatically pruned with the `min_match` parameter. The parameter `replace` is a boolean, indicating whether multiple treatment observations may match with the same control observation, which is `False` by default. Finally, the user has a choice of greedy (keyword argument: `"greedy"` or `"fast"`) or optimal matches (keyword argument: `"optimal"`, `"hungarian"`, `"kuhn"`, or `"munkres"`).

```
# Now we should match the graph
# By default: n_match=1, min_match=1, replace=False
mg.match(n_match=2, min_match=2, method="optimal")
```

Balance Assessment. The user can then compare the balance of the `input_data` and the `match_data`[8]. Note that `match_data` is a subset of `input_data`, plus a new column, `"match_group"`.

```
# Can get dataframes of input_data and match_data
input_df = mg.input_data.frame
match_df = mg.match_data.frame

```

[5] `filter_subgraphs` is particularly relevant for strata-based matching methods like CEM, which often filter strata based on the total number of observations (from each group), the ratio of treatment to control

[6] https://github.com/jackpotrykus/propensity-score-matching-thesis

[7] This is not always necessary, particularly in the case of CEM

[8] In the case of Exact matching edges, the `match_data` should consist of all subgraphs of order at least 2

```
# Assess balance improvement via the MatchingDataset.summary() method
input_balance = mg.input_data.summary()
match_balance = mg.match_data.summary()

# ... or import the balance functions themselves
# Let's compare the eCDF of scores between the input_data and the match_data
from matplotlib import pyplot as plt
import numpy as np
from matching.balance import ecdf

# Fit eCDFs to the scores in each dataframe
input_score_ecdf = ecdf(input_df["scores"])
match_score_ecdf = ecdf(match_df["scores"])

# Evaluate the eCDFs at 101 points between 0 and 1
xs = np.linspace(0, 1, 101)
input_score_ecdf_at_xs = input_score_ecdf(xs)
match_score_ecdf_at_xs = match_score_ecdf(xs)

# Plot them to compare
plt.plot(xs, input_score_ecdf_at_xs)
plt.plot(xs, match_score_ecdf_at_xs)
plt.show()
```

3.4 Comparison to Other Causal Inference Packages

Within the Python landscape, there are three existing packages capable of matching for causal inference: `DoWhy`, `pymatch`, and `dame-flame`. None of them directly use a graph data structure as their primary data structure for conducting the matching procedure, something which makes `matching` unique. `matching` also offers the simplest API for iterative matching procedures, as described in §3.3.

Below, I discuss the package in decreasing order of number of stars on its GitHub repo, as a proxy variable for its prevalence among Python data scientists alone (not necessarily a marker of quality!).

`DoWhy`. An "End-to-End Library for Causal Inference" developed by Microsoft (Sharma and Kiciman 2020), `DoWhy` supports matching on a variety of distance metrics, in addition to numerous other methods in the field of Causal Inference (e.g. instrumental variables), and can match the data, build a predictive model for outcomes, and estimate the treatment effect in one shot. The main drawback of this approach is that matching procedures[9] are rigid, "black-box" steps that occur as part of treatment effect estimation. The matching is neither easy to inspect nor flexibly alter without sub-classing `CausalEstimator`. In short, `DoWhy` is concerned with estimating treatment effects, and treats the matching procedure as a means to this end, not a process of note in itself. `matching` could be used to pre-subset the observational data before passing it to `DoWhy` for further analysis, but the fundamental aims of these packages differ: `DoWhy` is a high-level API for end-to-end causal inference, whereas `matching` enables low-level exploration of the matching process.

`pymatch`. Purpose-built by researchers working on an observational study (Miroglio 2022), `pymatch` supports only one distance metric, the propensity score, and the only matching algorithms on offer are "random" and "greedy"; no optimal matching is available. The package also lacks support iterative

[9] e.g. `dowhy.causal_estimators_propensity_score_estimator`

matching procedures. The codebase is short and succinct, and certainly works well for the specific use-case the researchers intended it for, but it is not feature-complete nor flexible enough to be considered a proper alternative to `MatchIt` or `matching`.

`dame-flame`. Developed by Duke's *Almost Matching Exactly Lab* (Gupta et al. 2021), `dame-flame` implements its two titular matching procedures: DAME (Liu et al. 2019) and FLAME (Wang et al. 2021). As discussed in §2.1.2, these methods extend CEM using machine learning, calculating a weight vector **w** indicating the relative importance of matching on each feature. These algorithms each run fast, and are easier to inspect than `DoWhy`. However, since these each seek to replicate CEM, and since these are the *only* matching methods provided by `dame-flame`, this means that `dame-flame` can only be used with discrete datasets; for users unwilling to coarsen/discretize their data, or users seeking a matching procedure other than DAME and FLAME, `dame-flame` is not an option. `matching` does not currently support these algorithms, but they are "in-the-works"; it can certainly be implemented as a new `matching.distance._Distance` subclass.

4 Experiments

4.1 Data Generation

Papers with data generation:

- Austin 2011
- Stuart, Lee, and Leacy 2013

5 Results

6 Discussion

7 Conclusion

Addenda

References

Austin, Peter C. (2011). "Optimal caliper widths for propensity-score matching when estimating differences in means and differences in proportions in observational studies". In: *Pharmaceutical Statistics* 10.2. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/pst.433, pp. 150–161. ISSN: 1539-1612. DOI: 10.1002/pst.433. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/pst.433.

Basu, Anirban, Daniel Polsky, and Willard G. Manning (June 2008). *Use of Propensity Scores in Non-Linear Response Models: The Case for Health Care Expenditures*. 14086. Publication Title: NBER Working Papers. National Bureau of Economic Research, Inc. URL: https://ideas.repec.org/p/nbr/nberwo/14086.html.

Dehejia, Rajeev H. and Sadek Wahba (Dec. 1999). "Causal Effects in Nonexperimental Studies: Reevaluating the Evaluation of Training Programs". In: *Journal of the American Statistical Association* 94.448, pp. 1053–1062. ISSN: 0162-1459, 1537-274X. DOI: 10.1080/01621459.1999.10473858. URL: http://www.tandfonline.com/doi/abs/10.1080/01621459.1999.10473858.

Garrido, Melissa M. et al. (Oct. 2014). "Methods for Constructing and Assessing Propensity Scores". In: *Health Services Research* 49.5, pp. 1701–1720. ISSN: 0017-9124, 1475-6773. DOI: 10.1111/1475-6773.12182. URL: https://onlinelibrary.wiley.com/doi/10.1111/1475-6773.12182.

Greifer, Noah (2022). "Assessing Balance". In: p. 22. URL: https://cran.r-project.org/web/packages/MatchIt/vignettes/assessing-balance.html.

Gupta, Neha R. et al. (Jan. 14, 2021). "dame-flame: A Python Library Providing Fast Interpretable Matching for Causal Inference". In: *arXiv:2101.01867 [cs]*. arXiv: 2101.01867. URL: http://arxiv.org/abs/2101.01867.

Hansen, Ben B (2006). "Bias Reduction in Observational Studies via Prognosis Scores". In: p. 29.

– (2008). "The Prognostic Analogue of the Propensity Score". In: *Biometrika* 95.2. Publisher: [Oxford University Press, Biometrika Trust], pp. 481–488. ISSN: 0006-3444. URL: https://www.jstor.org/stable/20441477.

Ho, Daniel et al. (June 14, 2011). "MatchIt: Nonparametric Preprocessing for Parametric Causal Inference". In: *Journal of Statistical Software* 42, pp. 1–28. ISSN: 1548-7660. DOI: 10.18637/jss.v042.i08. URL: https://doi.org/10.18637/jss.v042.i08.

Ho, Daniel E. et al. (2007). "Matching as Nonparametric Preprocessing for Reducing Model Dependence in Parametric Causal Inference". In: *Political Analysis* 15.3, pp. 199–236. ISSN: 1047-1987, 1476-4989. DOI: 10.1093/pan/mpl013. URL: https://www.cambridge.org/core/product/identifier/S1047198700006483/type/journal_article.

Iacus, Stefano M., Gary King, and Giuseppe Porro (Mar. 2011). "Multivariate Matching Methods That Are Monotonic Imbalance Bounding". In: *Journal of the American Statistical Association* 106.493, pp. 345–361. ISSN: 0162-1459, 1537-274X. DOI: 10.1198/jasa.2011.tm09599. URL: http://www.tandfonline.com/doi/abs/10.1198/jasa.2011.tm09599.

– (2012). "Causal Inference Without Balance Checking: Coarsened Exact Matching". In: *Political Analysis* 20.1, pp. 1–24.

Imai, Kosuke and David A van Dyk (Sept. 1, 2004). "Causal Inference With General Treatment Regimes". In: *Journal of the American Statistical Association* 99.467. Publisher: Taylor & Francis _eprint: https://doi.org/10.1198/016214504000001187, pp. 854–866. ISSN: 0162-1459. DOI: 10.1198/016214504000001187. URL: https://doi.org/10.1198/016214504000001187.

Khan, Arif et al. (Jan. 2016). "Efficient Approximation Algorithms for Weighted b-Matching". In: *SIAM Journal on Scientific Computing* 38.5, S593–S619. ISSN: 1064-8275, 1095-7197. DOI: 10.1137/15M1026304. URL: http://epubs.siam.org/doi/10.1137/15M1026304.

King, Gary and Richard Nielsen (Oct. 2019). "Why Propensity Scores Should Not Be Used for Matching". In: *Political Analysis* 27.4, pp. 435–454. ISSN: 1047-1987, 1476-4989. DOI: 10.1017/pan.2019.11.

URL: https://www.cambridge.org/core/product/identifier/S1047198719000111/type/journal_article.

Leacy, Finbarr P. and Elizabeth A. Stuart (Sept. 10, 2014). "On the joint use of propensity and prognostic scores in estimation of the average treatment effect on the treated: a simulation study". In: *Statistics in Medicine* 33.20, pp. 3488–3508. ISSN: 02776715. DOI: 10.1002/sim.6030. URL: https://onlinelibrary.wiley.com/doi/10.1002/sim.6030.

Liu, Yameng et al. (June 8, 2019). "Interpretable Almost Matching Exactly for Causal Inference". In: *arXiv:1806.06802 [cs, stat]*. arXiv: 1806.06802. URL: http://arxiv.org/abs/1806.06802.

Miettinen, O. S. (Dec. 1976). "Stratification by a multivariate confounder score". In: *American Journal of Epidemiology* 104.6, pp. 609–620. ISSN: 0002-9262. DOI: 10.1093/oxfordjournals.aje.a112339.

Miroglio, Ben (Apr. 22, 2022). *pymatch*. original-date: 2017-09-20T20:57:05Z. URL: https://github.com/benmiroglio/pymatch.

Munkres, James (Mar. 1957). "Algorithms for the Assignment and Transportation Problems". In: *Journal of the Society for Industrial and Applied Mathematics* 5.1, pp. 32–38. ISSN: 0368-4245, 2168-3484. DOI: 10.1137/0105003. URL: http://epubs.siam.org/doi/10.1137/0105003.

Rosenbaum, Paul R. (Dec. 1989). "Optimal Matching for Observational Studies". In: *Journal of the American Statistical Association* 84.408, pp. 1024–1032. ISSN: 0162-1459, 1537-274X. DOI: 10.1080/01621459.1989.10478868. URL: http://www.tandfonline.com/doi/abs/10.1080/01621459.1989.10478868.

Rosenbaum, Paul R. and Donald B. Rubin (1983). "The central role of the propensity score in observational studies for causal effects". In: *Biometrika* 70.1, pp. 41–55. ISSN: 0006-3444, 1464-3510. DOI: 10.1093/biomet/70.1.41. URL: https://academic.oup.com/biomet/article-lookup/doi/10.1093/biomet/70.1.41.

Sharma, Amit and Emre Kiciman (2020). "DoWhy: An End-to-End Library for Causal Inference". In: *arXiv preprint arXiv:2011.04216*.

Stuart, Elizabeth A. (Feb. 1, 2010). "Matching Methods for Causal Inference: A Review and a Look Forward". In: *Statistical Science* 25.1. ISSN: 0883-4237. DOI: 10.1214/09-STS313. URL: https://projecteuclid.org/journals/statistical-science/volume-25/issue-1/Matching-Methods-for-Causal-Inference--A-Review-and-a/10.1214/09-STS313.full.

Stuart, Elizabeth A., Brian K. Lee, and Finbarr P. Leacy (Aug. 2013). "Prognostic score–based balance measures can be a useful diagnostic for propensity score methods in comparative effectiveness research". In: *Journal of Clinical Epidemiology* 66.8, S84–S90.e1. ISSN: 08954356. DOI: 10.1016/j.jclinepi.2013.01.013. URL: https://linkinghub.elsevier.com/retrieve/pii/S0895435613001625.

Wang, Tianyu et al. (Feb. 4, 2021). "FLAME: A Fast Large-scale Almost Matching Exactly Approach to Causal Inference". In: *arXiv:1707.06315 [cs, stat]*. arXiv: 1707.06315. URL: http://arxiv.org/abs/1707.06315.

Zhu, Yeying, Jennifer S. Savage, and Debashis Ghosh (Sept. 25, 2018). "A Kernel-Based Metric for Balance Assessment". In: *Journal of Causal Inference* 6.2, p. 20160029. ISSN: 2193-3685, 2193-3677. DOI: 10.1515/jci-2016-0029. URL: https://www.degruyter.com/document/doi/10.1515/jci-2016-0029/html.