

Finding Common Support through Largest Connected Components and its
Implementation

by

Hongyuan Lu

B.S., Ohio State University, 2015

A REPORT

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Statistics
College of Arts and Sciences

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2019

Approved by:

Major Professor
Michael Higgins

Copyright

© Hongyuan Lu 2019.

Abstract

In an observational study, the average treatment effect may only be reliably estimated for a subset of units under which the covariate space of both treatment and control units overlap. This is known as the common support assumption. In this report, we develop a method to find a region of common support. The method is as follows. Given a distance function to measure dissimilarity between any two units with differing treatment statuses, we can construct an adjacency list by drawing edges between each pair of treated and control units that have distance no larger than some pre-specified threshold. Then, all connected components of the graph are found. Finally, a region of common support is found by obtaining the largest connected components (LCC) (e.g. the connected components with the most treated units) of this graph. We implement the LCC algorithm by using binary search trees to find all the connected graphs from sample data and sorting them by size. This algorithm requires $O(n^2)$ runtime and $O(n)$ memory (where n is the number of units in the observational study). We then create an R package implementing this LCC algorithm. Finally, we use our R package to compare the performance of LCC to that of other common support methods on simulated data.

Table of Contents

List of Figures	vi
List of Tables	vii
Acknowledgements	viii
Dedication	ix
Preface	1
1 Introduction	1
1.1 Causal Inference	3
1.2 Covariate Dissimilarity	5
1.3 Common Support and Existing Methods	6
1.4 Largest connected component for common support	8
2 Implementation and Algorithm	9
2.1 Introduction	9
2.2 Index and Data Storage	10
2.3 Variable Type	12
2.4 Adjacency List	13
2.5 Algorithm For LCC	16
2.6 Efficiency and Optimization	18
2.7 Improved Algorithm	19
3 Comparison and Testing	21

3.1	Current methods	21
3.1.1	Non-convex common support data	22
3.2	Non Convex Sample and Heterogeneous Data	24
3.3	Efficiency Bounds	25
3.4	Maxbox Method	29
3.5	BART	32
3.6	Optimal Match	35
3.7	Convex Hull	36
3.8	LCC Methods	38
3.9	Discussion	46
	Bibliography	47
A	Title for This Appendix	49
B	Title for This Appendix	50

List of Figures

2.1	10
2.2	11
2.3	14
2.4	15
2.5	19
3.1	26
3.2	27
3.3	28
3.4	30
3.5	32
3.6	33
3.7	34
3.8	35
3.9	37
3.10	38
3.11	39
3.12	40
3.13	41
3.14	42
3.15	43

List of Tables

2.1	Variable type correspondence between R and Python	12
3.1	results of efficiency bounds using true propensity score	25
3.2	results of efficiency bounds using well estimated propensity score	27
3.3	results of efficiency bounds using badly estimated propensity score	29
3.4	results of maxbox using true propensity score	31
3.5	results of maxbox using well estimated propensity score	31
3.6	results of bart for $\alpha = 0.1$ significance level	34
3.7	results of LCC using true propensity score	44
3.8	results of LCC using well estimated estimated propensity score	44
3.9	results of LCC using general euclidean distance	44
3.10	results of LCC using mahalanobis distance	45
3.11	results of LCC using the largest caliper	45

Acknowledgments

Enter the text for your Acknowledgements page in the `acknowledge.tex` file. The Acknowledgements page is optional. If you wish to remove it, see the comments in the `etdrtemplate.tex` file.

Dedication

Enter the text for your Dedication page in the `dedication.tex` file.¹²³ The Dedication page is optional. If you wish to remove it, see the comments in the `etdrtemplate.tex` file.

Chapter 1

Introduction

Consider an observational study with n treated units, each given either treatment or control. In this type of study, the inference on causal quantities of interest is ideal when the pre-treatment distributions of covariates are same for the treatment and control groups. A large difference in covariates between treatment and control groups may lead to a biased estimate of the average treatment effect when covariates are predictive of response. The solution to this problem is called statistical matching or modeling. A group of experiment units are included in the common support, where each unit has its close counterfactual.

Counterfactuals (Imbens and Rubin, 1997) are hypothetical unobserved outcomes. For example, for a treated unit, the counterfactual for that unit is the outcome if it had received the control. In causal inference, estimates of treatment effects are often made by using available outcomes to estimate counterfactuals. If the counterfactual models do not account for all important covariates, and accurately model the relationship between those covariates and response, the estimation of treatment effect could be biased. To prevent the above problem, it may be preferable to focus on estimation in an observational study for a subset of data under which common support holds. *Common support*—also known as covariate overlap—is a subset of the covariate space under which there exists both treatment and control units. In real world, it would be hard to find perfect counterfactuals, so we will let the common support to include all close counterfactuals.

Finding a region common support may be a tricky problem. Because every experiment unit carries some information, the exclusion of experiment units may lead to estimation bias. For example, if any treated unit is removed, matching estimators on the corresponding subset may no longer provide an unbiased estimation for the average treatment effect on the treated units (ATT). The method to find the common support should be able to efficiently create the common support with sufficient size to prevent the quantity of interest from changing substantially. For example, if we have 1000 treated units and 1000 control units, but the common support contains only 1 treated units and 2 control units, the size of the common support region may insufficiently small.

In this report, we investigate the Largest Connected Component (LCC) method for finding common support. The idea is as follows. Given a distance metric that measures the counterfactual dissimilarity of any two units pre-treatment covariates and a dissimilarity threshold ω , a graph is formed where each unit is a vertex in the graph and an edge is drawn between a treatment and control unit if and only if the dissimilarity is no larger than ω . LCC then forms a region of common support composed of the largest connected components of units in this graph. LCC offers a unique combination of computational efficiency, estimation precision, and flexibility of the sample data. For flexibility, LCC can find out convex or non convex common support regions for both convex and non convex data regions in $O(n^2)$ runtime. As the dissimilarity threshold increases, the quantity of interest on common support region approaches the quantity of interest on the original data set; however, this may introduce bias when complex relations exist between covariates, treatment, and response. In each component, every unit either received treatment or control, is connected to at least one another unit. The edge between two units indicates their dissimilarity is within the threshold ω and acceptable. The direct connection within a component is only between one treated unit and one control unit.

To code LCC efficiently, we perform a depth first binary search (DFS) [Cormen et al. \(2004\)](#) across the experiment units. This allows for increased computational efficiency for large thresholds; larger thresholds imply fewer “sparse paths” to compute when forming connected components. we have to consider fewer edges between units and have more indi-

rect connections and more experiment units are included. So we need to choose an optimal threshold. The memory space LCC takes is $O(np)$, where p is the number of covariates for each unit; when n is significantly larger than p , the memory space is $O(n)$. To facilitate the use of LLC by practitioners, we implemented the algorithm into an R package called *llc*. You can find it on my URL on github. Additionally, we provide the details to show how the LCC algorithm is implemented and a sample data simulation.

1.1 Causal Inference

Assume we have n experiment units indexed from 1 to n . Each unit is given either treatment or control. Let T_i denotes the treatment indicator for the i th unit. $T_i = 1$ if the i th unit receives the treatment, and $T_i = 0$ if the i th unit receives the control. We have n_0 units received control and n_1 units received treatment; $n = n_1 + n_2$. Suppose each unit has p observable covariates $x_i = (x_{i1}, x_{i2}, \dots, x_{ip}) \in R^p$. We assume the Neyman-Rubin potential outcomes (Brady et al., 2008) model for response. Let y_{i1} be the potential outcome for unit i if it received treatment and let y_{i0} be the potential outcome for unit i if it received control. Then y_{it} is the hypothetical outcome of unit i if that unit had received treatment t , where $t \in \{0, 1\}$. In this paper, we assume the potential outcomes not to be random. The observed response Y_i for unit i is can then be written

$$Y_i = y_{i1}T_i + y_{i0}(1 - T_i). \tag{1.1}$$

Inherent in this model is the stable unit treatment value assumption (SUTVA), which means the treatment status of a unit does not affect the response of any other unit. We further assume y_{it} is not only affected by random treatment status T_i but also significantly affected by the non-random covariates \mathbf{x}_i . The unit-level treatment effect for unit i is $\tau_i = y_{i1} - y_{i0}$. Here, τ_i is not observable, because no unit can receive both treatment and control. The average treatment effect ($ATE = E[Y_{i1} - Y_{i0}]$) measures the average treatment effect on all units for the whole population. The average treatment effect of treated ($ATT = E[Y_{i1} - Y_{i0} | T_i =$

1]) measures the average treatment effect on the treated units. Here, the expectation is computed a randomly sampled unit ι from the population. Most often in observational studies, the quantity of interest is the ATT.

Unbiased estimation of the ATE requires the following two assumptions ([Mahmood et al.](#)):

Assumption 2.1 (Unconfoundedness):

$$T_i \perp (y_{i1}, y_{i0}) \mid \mathbf{x}_i \tag{1.2}$$

Assumption 2.2(Common Support for ATE): For all \mathbf{x}_i ,

$$0 < P(T_i = 1 \mid \mathbf{x}_i) < 1 \tag{1.3}$$

The assumptions hold over all potential realizations for $(y_{i1}, y_{i0}, T_i, \mathbf{X}_i)$ in the population ([Mahmood et al.](#)). Assumption 2.1 indicates that for given pretreatment covariates, the treatment indicator for any unit is independent of its potential outcomes. Assumption 2.2 indicates the probability to receive treatment is between 0 and 1. The common support region is where the covariates overlap. It ensures that the ATE can be well defined. The unconfoundedness and overlap assumptions constitute a property called strong ignorability of assignment ([Rosenbaum and Rubin, 1983](#)).

When estimating the ATT, Assumption 2.1 can be restricted to realizations in which $T_i = 1$ and Assumption 2.2 can be replaced by Assumption 2.3.

Assumption 2.3(Common Support for ATT): For all \mathbf{x}_i :

$$0 \leq P(T_i = 1 \mid \mathbf{x}_i) < 1 \tag{1.4}$$

The sample ATE and sample ATT have the formulas as following respectively:

$$SATE = \frac{1}{n} \sum_{i=1}^n (y_{i1} - y_{i0}) \tag{1.5}$$

$$SATT = \frac{1}{n_1} \sum_1^{n_1} (y_{i1} - y_{i0}) \quad (1.6)$$

In this paper, we focus on finding the estimation of the sample average treatment effect on treated (SATT).

1.2 Covariate Dissimilarity

Our method for estimating SATT requires a dissimilarity measure between units that should be small when two units have similar values of prognostically important covariates. For unit i with covariate vector \mathbf{x}_i and unit j with covariate vector \mathbf{x}_j , where $\mathbf{x}_i, \mathbf{x}_j \in R^p$, let w_{ij} denote the dissimilarity between unit i and unit j . Now we need to choose the dissimilarity measure w_{ij} such that w_{ij} is small when \mathbf{x}_i and \mathbf{x}_j have similar values and w_{ij} is large when $\mathbf{x}_i, \mathbf{x}_j$ are very different. There are many different possible choices for the dissimilarity measure. We now outline a few of these.

Our first choice is the standard euclidean distance. To get the standardized Euclidean distance, first we have to standardize the covariates. Let \hat{X} denote the standardized value for covariate X . Let s denote the standard deviation for covariate X . The standardized value is

$$\hat{X} = (X - \bar{X})/s \quad (1.7)$$

Then we will apply General Euclidean distance on the standardized covariates. Let's denote the standardized covariates as X . The general euclidean distance between \mathbf{x}_i and \mathbf{x}_j is

$$w_{ij} = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \dots + (x_{ip} - x_{jp})^2} \quad (1.8)$$

Besides standardized Euclidean distance, Mahalanobis distance is also popular for measuring distance between two units. We can also use the propensity scores to measure distances. A propensity score for unit i is the probability that this unit receives treatment given covariate \mathbf{x}_i . It is written as $e(\mathbf{x}_i) = P(T_i = 1|\mathbf{x}_i)$. If propensity score $e(\mathbf{x}_i)$ of unit i is strongly associated with its covariates \mathbf{x}_i , then the difference of two propensity scores may represent

the differences between covariates of two different units.

The choice of dissimilarity measure is flexible and should serve the researchers interest, but it should always satisfy two conditions. First, for any dissimilarity w_{ij} , between unit i and j , $w_{ij} \geq 0$. Second, the dissimilarity needs to satisfy the triangle inequality: for any three units i, j, k , $w_{ij} + w_{jk} \leq w_{ik}$. Those two conditions are very important when using LCC methods. Considering all units to be points and dissimilarity measures to be edges among directly connected units, the importance of the two conditions can be easily understood graphically.

1.3 Common Support and Existing Methods

In this paper, several methods to find a common support region are compared. Those methods will be compared on several aspects. First we will compare their functionality for large observational studies, especially when the number of experiment units n is large. Some current common support methods may require prohibitive computation to be applied to datasets with large n . Second we will compare the flexibility of these methods, both in terms of the size of the region of common support and how these methods can detect non-convex regions of common support. A too small size of common support region may lose significant information carried by the experiment sample, insert a bias and potentially increase the variance of estimation. A too large common support may be lack of the covariate overlap. We will test these methods on sample data with non-convex regions for covariates. Finally, we will test the robustness of estimation of treatment effects after finding the common support methods. We will test whether these methods work on good and bad propensity estimations, and whether these methods can use distance measures other than propensity scores, such as standard Euclidean, Mahalanobis or largest caliper ([Mahmood et al.](#)) distances. This is important when a good estimate of the propensity score is unavailable.

[Crump et al. \(2009\)](#) identified the study population by finding the optimal subsamples which minimize the efficiency bound for the variance of average treatment effect. For our sample data, it describes a simple rule to remove all units from the study population with

estimated propensity scores outside the range $[0.1, 0.9]$. [Rosenbaum \(2012\)](#) formed an optimal subset of the sample to identify the common support where one chooses the upper bound on the maximum number of treated units that can be removed by optimal matching. Each matchable treatment unit and control unit are paired, and the average distance within pairs should be minimized. ([Fogarty et al., 2016](#)) formed a common support region by constructing a maxbox for important covariates. The maxbox gives a condition for each experiment included in the common support. We will use estimated propensity to create the maxbox area. For maxbox, the experiments units should have propensity score between 0.1 and 0.9. And we will maximize X_U and minimize X_L so as well as maximizing the common support area such that for all units in common support $X_L \ll X \ll X_U$. It provides a rectangular regions of the covariate space for common support. ([Hill and Su, 2013](#)) identified common support by employing Bayesian Additive Regression Trees (BART).

We will test these common support methods against our new method for finding common support—largest connected components—on several aspects. First we will compare their functionality for large observational studies, especially when the number of experiment units n is large. Some current common support methods may require prohibitive computation to be applied to datasets with large n Second we will compare the flexibility of these methods, both in terms of the size of the region of common support and how these methods can detect non-convex regions of common support. A too small size of common support region may lose significant information carried by the experiment sample, insert a bias and potentially increase the variance of estimation. A too large common support may be lack of the covariate overlap. We will test these methods on sample data with non-convex regions for covariates. Finally, we will test the robustness of estimation of treatment effects after finding the common support methods. We will test whether these methods work on good and bad propensity estimations, and whether these methods can use distance measures other than propensity scores, such as standardiex Euclidean, Mahalanobis or largest caliper ([Mahmood et al.](#)) distances. This is important when a good estimate of the propensity score is unavailable.

1.4 Largest connected component for common support

Largest connected components (LCC) is a recently developed method for finding common support (Mahmood et al.). The details of LCC method could be find in paper *Finding Common Support through Largest Connected Components*(Mahmood et al.). This paper is mainly focused on the implementation of LCC method and the result of comparison with other methods. So I will give only a general introduction of LCC methods. For LCC, we view our data as a bipartite graph $G = (V = (V_1, V_0), E)$, where V_1 and V_0 are sets of vertices and E is a set of edges. Each unit receiving treatment corresponds to a vertex in V_1 and each unit receiving control corresponds to a vertex in V_0 . For each pair of vertices $i \in V_1$ and $j \in V_0$, there is a unique edge $e_{ij} \in E$. That is G is a complete bipartitie graph with $|V_1| = n_1$ and $|V_0| = n_0$, where n_1 is the number of treated units and n_0 is the number of control units. Each edge $e_{ij} \in E$ has a weight $w_{ij} \geq 0$ representing the dissimilarity between units i and j , where $w_{ij} \geq 0$.

We then construct a *bottleneck subgraph* $G_\omega^* = (V, E_\omega)$ for threshold ω where an edge $ij \in E_\omega$ if and only if that edge has weight $w_{ij} \leq \omega$. The LCC region of common support is then comprised of the largest connected components of this bottleneck subgraph. A connected component is a subgraph in which each observational unit is connected to at least one other observational unit. The size of our common support region can be adjusted by changing the value of threshold ω , larger ω correspond to larger regions of common support.

Compared to available methods for finding common support, we show LCC offers great computational efficiency especially when the number of experiment units n is large. Second LCC allows flexibility for the shapes permitted for common support, including non-convex regions. Third, unlike some other methods which are highly dependent on the estimation quality of propensity scores, the LCC method offers robustness estimation for many different types of dissimilarity measures.

Chapter 2

Implementation and Algorithm

2.1 Introduction

As mentioned by [Mahmood et al.](#), the LCC algorithm requires a dissimilarity measure w_{ij} and a dissimilarity threshold ω . The general idea of LCC algorithm is simple. Suppose we have n_1 treated units and n_2 control units. The LCC algorithm is as follows [Mahmood et al.](#):

Step 1: Consider all the suits as bipartite graph $G = (V = (V_1, V_0), E)$ where V_1 is the set of all treated units and V_0 is the set of all control units.

Step 2: Find the acceptable matches: Find all pairs of units with opposite treatment statuses that have dissimilarity less than threshold ω .

Step 3: Form a bottleneck graph where edges join acceptable matches.

Step 4: Find the set of connected components in the bottleneck graph.

Step 5: Identify the largest connected components. Ensure that the connected components have sufficiently many treated and control units to estimate treatment effects within that connected component with little bias.

Step 6: Select all the units that are in largest connected components to obtain a region of common support.

We will offer algorithms to implement this idea in rest of this chapter.

2.2 Index and Data Storage

Although R is very useful in data handling, it is not very friendly for coding the algorithm. First, R takes long time, especially when running loops. Second, R takes a large amount of memory. For any parameter input into a user built R function, a new copy of the parameter is created. A copy of a large parameter takes large memory space and slows down algorithm.

To mitigate this problem, we use Python to implement the algorithm. This causes another issue—R indexes vectors differently than other lower level languages like C, C++, Java, or Python.

(R index)	X1	X2	X3	Treatment	Response
1	1	
2	1	
3	1	
4	1	
5	1	
6	1	
7	1	
8	1	
9				1	
10				1	
11				0	
12				0	
13				0	
14				0	
15				0	
16				0	
17				0	
18				0	
19				0	
20				0	

Figure 2.1: *Data Structure*

Suppose we have a data as in Figure 2.1. Let X_1, X_2, X_3 be the three covariates in our sample data.

For this dataframe in figure 2.1, we have 20 units. Ten of them receive treatment and

are indexed from 1 to 10. Ten of them receive control and are indexed from 11 to 20. Then we can separate the treated and control units into two groups and input them into Python separately. For a vector, the first element in R is indexed 1, and in Python, it is indexed 0. The last element index in R is the vector length, and in Python, it is the length minus 1. For convenience, we will use the Python index during the algorithm and we will map the result back to R index after algorithm.

Let's import covariates of treated units and control units into two groups (a dictionary) in Python. In Python, treated units are indexed from 0 to $n_1 - 1$, and control units are indexed from 0 to $n_2 - 1$. We do not need to save either the Python indexes or R indexes now, because Python and R indexes are automatically created. This saves memory cost. The input data structure is in Figure 2.2.

Treated				Control					
(python index)	(R index)	X1	X2	X3	(python index)	(R index)	X1	X2	X3
0	1	0	11
1	2	1	12
2	3	2	13
3	4	3	14
4	5				4	15			
5	6				5	16			
6	7				6	17			
7	8				7	18			
8	9				8	19			
9	10				9	20			

Figure 2.2: *Inputted data*

We now have two vectors in Python of three elements. Each element in each vector contains a covariate vector, one for each of the three covariates. For our sample data, the vectors of vector in R is a dataframe and the vectors of vector in Python is a dictionary. In general, a dataframe from R is a dictionary in Python. For our sample data in Figure 2.1, the column names in R are the dictionary keys in Python; they are $\mathbf{X}_1, \mathbf{X}_2$ and \mathbf{X}_3 . We now

have two dictionaries for treated units and control units in Python. The associated content for each key is a vector of the corresponding covariate vector. In the treated dictionary, the vector for key $X1$ contains $x_{11}, x_{21}, \dots, x_{n_1 1}$ in R dataframe. In the control dictionary, the vector for key $X1$ contains $x_{(n_1+1)1}, x_{(n_1+2)1}, \dots, x_{(n_1+n_2)1}$ in the R dataframe. Now we can map the corresponding index between R and Python—if a unit is indexed i in the Python treated dictionary, it is the row $i + 1$ in the R dataframe, and if a unit is index i in the Python control dictionary, it is the row $i + n_1 + 1$ in R dataframe. These two dictionaries take $O(np)$ memory space. When n is significantly larger than p , it takes $O(n)$ memory space. Because we already have treated units in upper rows in our R dataframe.

2.3 Variable Type

In most of the computer languages including R and Python, the variable type is very important. When a variable is imported from R to Python or vice versa, we can hardly do any implementation before knowing the variable type. The following table shows corresponding variable types between R and Python. One important aspect is that, for a dataframe in R, its corresponding variable type in Python is a dictionary. Our `11c` package takes dataframes or matrices as parameters. The column names in the dataframe are the keys in the dictionaries. If the input is a matrix, we will assign it column names and still get dictionaries in Python. Table 2.3 shows the variable type correspondence between R and Python.

R	Python
Multi-element vector	List
List of multiple types	Tuple
Single-element vector	Scalar
Named list(such as dataframe)	Dictionary
Matrix/Array	NumPy ndarray
Function	Python function
Raw	Python bytearray
NULL, TRUE, FALSE	None, True, False

Table 2.1: Variable type correspondence between R and Python

2.4 Adjacency List

We check the edges between every treated unit and every control unit according to the distance w_{ij} and threshold ω then store them in adjacency lists. We have defined the edge in a bottleneck subgraph such that, if a distance between a treated unit and a control unit is no larger than the threshold ω , there is an edge connecting them. The adjacency lists will be implemented using dictionaries. We use two dictionaries to store all the adjacency lists. One dictionary is for treated units and the other is for control units. In treated dictionary, the keys are Python indexes of the treated units. For each key the corresponding content is a list of indexes for all control units connected to that treated units. For control dictionary, it is the opposite. These two dictionaries may take $O(n^2)$ memory space, but can take much less if the threshold is sufficiently small.

For example suppose edges are shown as in Figure 2.3. Figure 2.4 is our adjacency list. Figure 2.3 uses the R index, and the adjacency lists in Figure 2.4 uses the Python index. We will perform a mapping to get R indexes after the algorithms are finished.

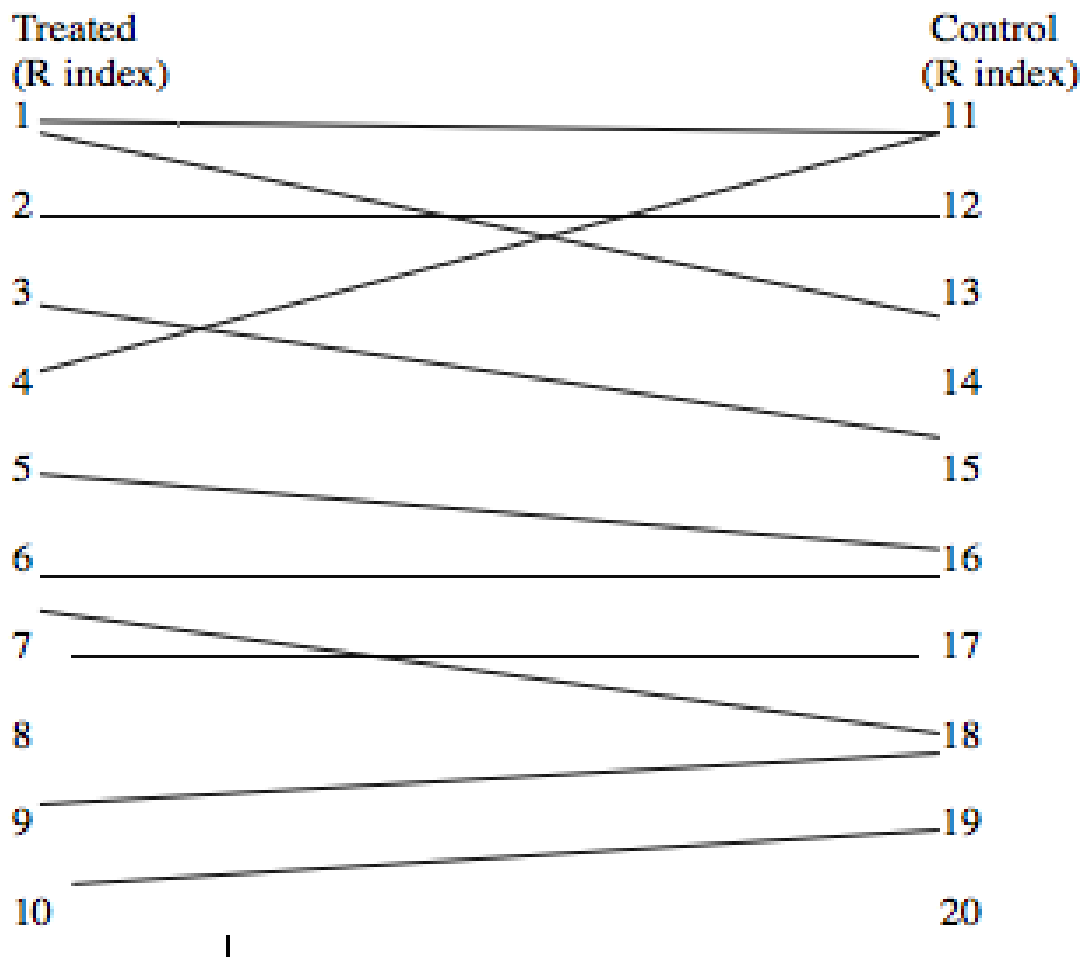


Figure 2.3: *Connected graph*

Treated:		Control	
Keys()	lists	keys	lists
0	[0,2]	0	[0,3]
1	[1]	1	[1]
2	[4]	2	[0]
3	[0]	3	[]
4	[5]	4	[2]
5	[5,7]	5	[4,5]
6	[6]	6	[6]
7	[]	7	[5,8]
8	[7]	8	[9]
9	[8]	9	[]

Figure 2.4: *Adjacency list for the connected graph in Figure 2.3*

2.5 Algorithm For LCC

We now find the largest connected subgraphs. We use a Depth First Search (DFS) with two stacks. One stack is for treated units, the other is for control units. There is nothing too complicated to find the largest connected graphs using DFS as we now have an efficient way to store data. The only thing unique is that two stacks for both treated and control units are used in depth first search.

In our data structure, all child nodes for a treated root are controls units and vice versa; no treated unit can be directly connected to any other treated unit, and no control unit can be directly connected to any other control unit. For control units, it is vice versa. That's why we have two stacks. If we find a unit is connected to a graph, we call this unit encountered and mark it through two Boolean vectors.

The Boolean vectors take advantage of our well indexed Python data storage. Without using Boolean vectors, if we put the indexes of detected treated units into a list called "trt-encountered", each time we encounter another treated unit, we need to check if it is in the trt-encountered list. This takes $O(n_1)$ runtime, where n_1 is the number of treated units in total. Instead, we use a list of Boolean variables of length n_1 and indexed from 0 to $n_1 - 1$ to mark the encountered treated units. We will set the default values of every elements in these list to be false. If a treated unit with index i is encountered for the first time, we mark it by setting the element in i th position in the Boolean vector to be true. Then to check if a treated unit j has been encountered before, we just need to check the j th position in the Boolean vector. If the value is true, it has been encountered, and if the value is false, it has not. Each check takes $O(c)$ time instead of $O(n_1)$ and $O(n_0)$ for treated and control units respectively. The two Boolean vectors take exactly n bits memory space.

Now we start our DFS. We will start from any undetected treated node and use the corresponding adjacency list to find all its undetected child control units, mark them as encountered, and put them into the control stack. Then we pop out the control unit on top of the control stack, find all its child treated units, mark them as encountered, and put them on top of the treated stack. Then we pop out the unit on top of the treated stack

and repeat the algorithm until the treated stack is empty. Then we pick another unmarked treated units and repeat the algorithm until all treated units are marked. The algorithm is as follows:

Step 1: Calculate all distances for each pair of treatment node and control node to build the adjacency lists.

Step 2: Choose any undetected treated unit, mark it as encountered, and put it on top of the treatment stack.

Step 3: Pop out out the node on top of the treatment stack, find all its unmarked control child nodes using adjacency lists, mark them as encountered, and put them on top of the control stack. Then pop out the node on top of the control stack, find all its unmarked child treated nodes and put them into the treatment stack.

Step 4: If the control stack is empty, we go back to Step 2. If the treated stack is empty, the nodes in the previous Step 2 and following Step 3 form a connected graph. Record the graph.

Step 5: Repeat step 2 and 3 until all treated units are marked.

Step 6: Output a list of list pairs from Python.

Each pair of lists in the output is a connected component units. In each pair, the first list contains the Python indexes of treated units in the cluster, the second list contains the Python index of the control units in the cluster. We map the Python indices back into R indices to obtain connected components, each having at least one treated unit. All control units not in any connected component do not have a “sufficiently close” treated unit are not connected directly to any other treated units.

2.6 Efficiency and Optimization

Suppose our data frame has n nodes, with half units received treatment and half units received control. DFS takes algorithm time of $O(n + m)$, where n is the number of units and m is the number of edges. Our LCC algorithm takes $O(n^2)$ algorithm time. Calculating all distances between each treated node and control node to create adjacency lists takes $O(n_1 n_2)$ algorithm time, which could be slow. For a sample data of 100,000 units with half treated and half control, we will have to calculate 2.5 billion distances. Second the adjacency lists may require a lot of memory to store if the threshold is too large—up to $O(n^2)$ memory space. If each treatment unit is connected with each control unit, then for these 100,000 units our adjacency lists have 100 billion total length. A question then becomes, how can we calculate fewer distances and use a less memory?

There are two improvements we can make. First the entire adjacency lists carries redundant information. For example, if we know nodes A and C are connected, and nodes B and C are connected, then calculating distance between nodes A and B would be unnecessary; nodes A and B are connected through C. In other words, we should take advantage of the indirect connections among the nodes; if nodes A and B have been encountered and have been determined to belong to the same connected component, it is unnecessary to compute the distance between A and B. Additionally, if a connected component that has been found contains node A but does not contain node B, then A and B are not connected in the bottleneck subgraph, and hence, it is unnecessary again to compute the distance between A and B. Figure 2.5 shows the idea graphically. Second the adjacency list itself is not necessary. The DFS algorithm in Section 2.5 uses every connection in the adjacency lists only once. Hence, we can calculate the distance and access the connection while searching the depth first tree. By making these two improvements, this we can save the $O(n_1 n_2)$ memory space required for the adjacency lists and calculate fewer distances.

We now observe can see that to build the adjacency list, Calculating distance between every treated node and control node is unnecessary. Instead we only calculate the distance between the current root node and every undetected node who received the opposite treat-

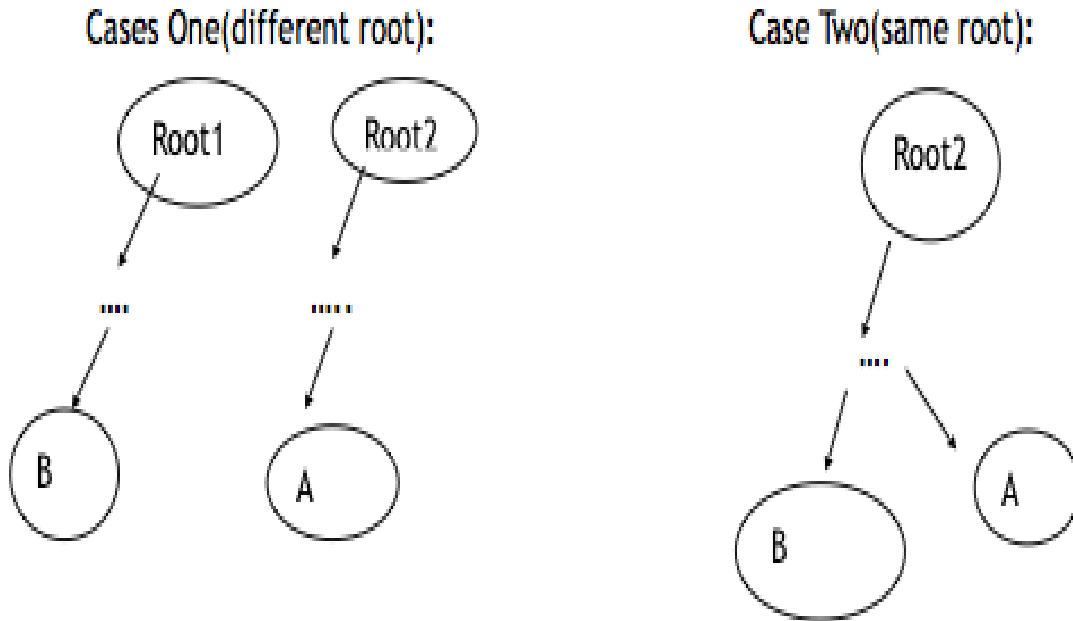


Figure 2.5: *Connected graph*

ment condition during the DFS. By doing this we can save the $O(n_1n_2)$ memory space for adjacency lists and calculate less distances.

2.7 Improved Algorithm

We implement the ideas in Section 2.6 to obtain an improved algorithm:

Step 1: Start from any unmarked treatment node as our root. Mark it as encountered and push it on top of the treatment stack.

Step 2: Pop out the treatment node on top of the treatment stack and calculate the distance between it and every unmarked control nodes. If the distance is smaller than the threshold ω , mark the control node as encountered and put on top of the control stack. Then, pop out out the node on top of the control stack. Calculate distances between this node and each unmarked treated nodes. If the distance is smaller than the threshold ω , we mark the treated node and push it on top of the treated stack.

Step 3: Repeat Steps 2 and 3 until treatment stack or control stack is empty. If treatment

stack is empty, record all nodes we encountered from previous Step 1 and following Step 2. These nodes comprise a connected graph. If control stack is empty, go to step 2.

Step 4: Repeat previous steps until there is no unmarked treatment node.

For this algorithm, instead of calculating all distances among all treated units and control units, we calculate fewer distances. The larger the dissimilarity threshold ω , the more indirect connections there will be among all nodes, and the fewer distances need to be calculated. In other words, each unit could be attached to a cluster through only one direct connection.

Also we do not need to use memory to store the adjacency list, we only record one distance at a time in the algorithm. The memory space to store adjacency list could be as large as $O(n^2)$, while it only takes memory space of $O(c)$ to store a single distance.

To assess the computational gain from the improved algorithm, we apply it to a toy data example containing 230 treated units and 770 control units, where dissimilarity is measured using the Euclidean distance. The original algorithm required us to calculate 177100 distances. Using this improved algorithm, when $\omega = 0.6$, we are required to compute 173,375 distances, for $\omega = 0.6$ we compute 144,585 distances, and for $\omega = 1.2$ we compute 96,327 distances. We see a clear tendency that, as the threshold increase, the distances need to be calculated decreases.

Finally we convert the Python indexes back into R indexes. For example in Figure 2.1, for the largest graph, we get a treatment list of Python indexes of $\{0, 3\}$, and control list of Python indexes of $\{0, 2\}$. Using the mapping in Figure 2.2, we have the R index of $\{1, 3, 11, 13\}$ as our largest connected graph. These are the rows of $\{1, 3, 11, 13\}$ in the dataframe from R, and they compose the largest connected graph.

Chapter 3

Comparison and Testing

3.1 Current methods

There are several existing methods to generate common support regions, such as Efficiency Bounds (Crump et al., 2009), Optimal Match (Rosenbaum, 2012), BART (Hill and Su, 2013), Maxbox (Fogarty et al., 2016) and Convex hull (King and Zeng, 2006).

These methods may generate common support regions with well balanced propensity scores or covariates, but can also perform badly for some specific conditions, or consume too much time for a large data sample size. Generally speaking, Efficiency Bounds (Crump et al., 2009) is highly dependent on the quality of the estimation on propensity scores. Optimal Matching (Rosenbaum, 2012) is time consuming; using the Mahalanobis distance, this algorithm required hours of runtime on my computer for 10000 data points with 3 covariates. BART (Hill and Su, 2013) can work on non-convex datasets and can take advantage of using observable covariates, but attempts to incorporate additional units outside of the region of common support to perform estimation may create bias for certain types of response schedules. The Maxbox method (Fogarty et al., 2016) only gives a rectangular common support region; it may struggle to generate a sufficiently large common support region for non-convex data. Additionally, The convex hull method (King and Zeng, 2006) may struggle to find an acceptable not generate a good common support region when that

region is non-convex.

3.1.1 Non-convex common support data

We now build a simulated data set with a non-convex region of common support to test the above methods. Let Z be the treatment indicator. Let $z=1$ if the unit receives treatment and $z=0$ if the unit receives control. A propensity score $e(\mathbf{x})$ is the probability that a unit is assigned with treatment given the covariates of \mathbf{x} , written as $e(\mathbf{x}) = P(z = 1|\mathbf{x})$.

We generate a dataset with 10,000 units. Each unit i is composed of three covariates $\mathbf{x}_i = (x_{i1}, x_{i2}, x_{i3})$ where \mathbf{x}_i is generated by multivariate normal distribution $N(\mu, \Sigma)$, where

$$\mu = (0, 0, 0), \quad \Sigma = \begin{vmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{vmatrix}$$

The true propensity scores is derived by

$$e(\mathbf{x}_i) = \begin{cases} \frac{4 - x_{i1}^2 - x_{i2}^2}{3.5}, & 0 < \frac{4 - x_{i1}^2 - x_{i2}^2}{3.5} < 1, \\ 1, & \frac{4 - x_{i1}^2 - x_{i2}^2}{3.5} \geq 1, \\ 0, & \frac{4 - x_{i1}^2 - x_{i2}^2}{3.5} \leq 0. \end{cases} \quad (3.1)$$

We use the true propensity scores to generate our treatment indicator vector $z = (z_1, z_2, \dots, z_n)$ using Bernoulli distribution, where $P(z_i = 1) = e(\mathbf{x}_i)$.

And our response is $Y = (y_1, y_2, \dots, y_n)$, where $y_i = \frac{1}{(x_{i1}^2 + x_{i2}^2 + x_{i3}^2)} + 2 * z_i$. So in our sample data set, the true treatment effect on every treated unit is 2.

And we also generate a well estimated propensity score using logistic linear model where $logit[\hat{e}(\mathbf{x}_i)] = \beta_1 * (x_{i1} + x_{i2} + x_{i3})^2 + \beta_2 x_{i1}^2 + \beta_3 * x_{i2}^2 + \beta_4 * x_{i3}^2$.

We will use the above sample to get the the common support region using different methods and our LCC method. In each method we will plot the common support region graph for x_1 vs x_2 to compare common support and original sample data set. The blue point in the

graph indicates a unit received control and a red point on the graph indicates a unit received treatment. We will also give the the back to back histogram and boxplot to check the balance of propensity scores.

Homogeneous treatment effect data model

Now let's build the homogeneous testing data set with a non-convex region of covariates and test the above methods. Let Z be the treatment indicator. Let $z=1$ if the unit receives treatment and $z=0$ if the unit receives control. A propensity score $e(\mathbf{x})$ is the probability that a unit is assigned with treatment given the covariates of \mathbf{x} , written as $e(\mathbf{x}) = P(z = 1|\mathbf{x})$.

Suppose we have a data set composed of 10000 units. Each unit is composed of three covariates x_1, x_2 and x_3 . So we have vectors such that $\mathbf{x}_i = (x_{i1}, x_{i2}, x_{i3})$ where \mathbf{x}_i is generated

by multinormal distribution (μ, σ) , where $\mu = (0, 0, 0)$ and $\sigma = \begin{vmatrix} 2 & 0 & 2 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{vmatrix}$

Our true propensity scores is derived by

$$\begin{aligned} e(\mathbf{x}_i) &= \frac{4-3*x_{i1}^2-x_{i2}^2}{3.5} && \text{if } 0 < \frac{4-x_{i1}^2-x_{i2}^2}{3.5} < 1 \\ e(\mathbf{x}_i) &= 0 && \text{if } \frac{4-x_{i1}^2-x_{i2}^2}{3.5} \leq 0 \\ e(\mathbf{x}_i) &= 1 && \text{if } \frac{4-x_{i1}^2-x_{i2}^2}{3.5} \geq 1 \end{aligned}$$

We use the ture propensity scores to generate our treatment indicator vector $z = (z_1, z_2, \dots, z_n)$ using binary distribution, where $P(z_i = 1) = e(\mathbf{x}_i)$. And our response is $Y = (y_1, y_2, \dots, y_n)$, where $y_i = \frac{1}{(x_{i1}^2+x_{i2}^2+x_{i3}^2)} + 2 * z_i$. So in our sample data set, the true treatment effect on every treated unit is 2.

And we also generate a well estimated propensity score using logistic linear model where

$$\text{logit} [\hat{e}(\mathbf{x}_i)] = \beta_1 * (x_{i1} + x_{i2} + x_{i3})^2 + \beta_2 x_{i1}^2 + \beta_3 * x_{i2}^2 + \beta_4 * x_{i3}^2.$$

We will use the above sample to get the the common support region using different methods and our LCC method. In each method we will plot the common support region graph for x_1 vs x_2 to compare common support and original sample data set. The blue point in the graph indicates a unit received control and a red point on the graph indicates a unit received treatment. We will also give the the back to back histogram and boxplot to check the balance

of propensity scores.

3.2 Non Convex Sample and Heterogeneous Data

Now let's also create a heterogeneous Data sample. Let's use the same covariates of the homogeneous model. So In this heterogeneous model $\mathbf{x}_i = (x_{i1}, x_{i2}, x_{i3})$ where \mathbf{x}_i is generated

by multinormal distribution (μ, σ) , where $\mu = (0, 0, 0)$ and $\sigma = \begin{vmatrix} 2 & 0 & 2 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{vmatrix}$

Let's also use the same true propensity scores in the homogeneous model.

$$\begin{aligned} e(\mathbf{x}_i) &= \frac{4-3x_{i1}^2-x_{i2}^2}{3.5} && \text{if } 0 < \frac{4-x_{i1}^2-x_{i2}^2}{3.5} < 1 \\ e(\mathbf{x}_i) &= 0 && \text{if } \frac{4-x_{i1}^2-x_{i2}^2}{3.5} \leq 0 \\ e(\mathbf{x}_i) &= 1 && \text{if } \frac{4-x_{i1}^2-x_{i2}^2}{3.5} \geq 1 \end{aligned}$$

Let's use the same well estimated propensity scores in the homogeneous model, such that

$$\text{logit}[\hat{e}(\mathbf{x}_i)] = \beta_1 * (x_{i1} + x_{i2} + x_{i3})^2 + \beta_2 x_{i1}^2 + \beta_3 * x_{i2}^2 + \beta_4 * x_{i3}^2.$$

The only difference between the homogeneous model and heterogeneous model is that, in the heterogeneous model the response denoted as \hat{y}_i , for each i $\hat{y}_i = \frac{1}{x_{i1}^2+x_{i2}^2+x_{i3}^2} + 2x_{i1}z_i + 3x_{i2}z_i + 2z_i$, where z_i is the treatment indicator. In the heterogeneous model, let's calculate the true treatment effect on the treated(ATT) for each common support generated from different methods. The true treatment effect is the average of potential outcome for units given treatment minus the average of potential outcome for units given control. For different common supports, the common support graphs are same for both homogeneous model and heterogeneous model because they are achieved from either same covariates or propensity scores except for the BART. However, unlike using the homogeneous model during which all the true ATTs are 2, using the heterogeneous model the true ATTs are different for each different common support. The true ATT for the entire model is 1.945225. For heterogeneous model, if the true ATT of common support is far away from the true ATT of the data sample, then the common support fails to represent the the sample data. We will compare these ATTs

with the estimated ATTS using different matching methods.

3.3 Efficiency Bounds

Efficiency bounds [Crump et al. \(2009\)](#) generates common support regions highly dependent on propensity scores. From Dealing with limited overlap in estimation of average treatment effects [Crump et al. \(2009\)](#), it is a good start to use range $[0.1,0.9]$ as the boundary for the propensity scores. Any unit with a propensity score smaller than 0.1 or larger than 0.9 will be excluded from the common support region. This causes another problem, when the propensity scores are distributed not dispersed enough, the $[0.1,0.9]$ can not be applicable. To solve this, the propensity scores could be cut according to the percentiles. We can cut the highest 10% propensity scores and lowest 10% propensity scores. During our test, we will first use the the real propensity scores then estimated propensity scores with boundaries to achieve the common support region. Now let’s first cut on true propensity socres. Figure 3.1 is the graph of the common support compared with the original data set for x_1 versus x_2 cutting on real propensity scores, where each blue point is a treated unit and each red point is a control unit. This figure is for both heterogeneous model and homogeneous model. The common support consists of 4029 experiment units with 2188 units received treatment.

From Figure 3.1 we know that Efficiency Bounds [Crump et al. \(2009\)](#) methods works on the non-convex sample data set and generate a non-convex common support region.

Efficiency Bounds(True Propensity Score)		
<i>Common Support</i>		
Running Time	Common Support Size	# Treated Units
Instant	4029	2188
<i>Estimated ATT</i>		
	GenMatch	Propensity Match
Estimated ATT	2.0032	2.0038
AI-SE	0.0028003	0.0096942
P-Value	<2.22e-16	<2.22e-16

Table 3.1: results of efficiency bounds using true propensity score

For the homogeneous model, the estimated ATT is 2.0032 using genMatch and 2.0038

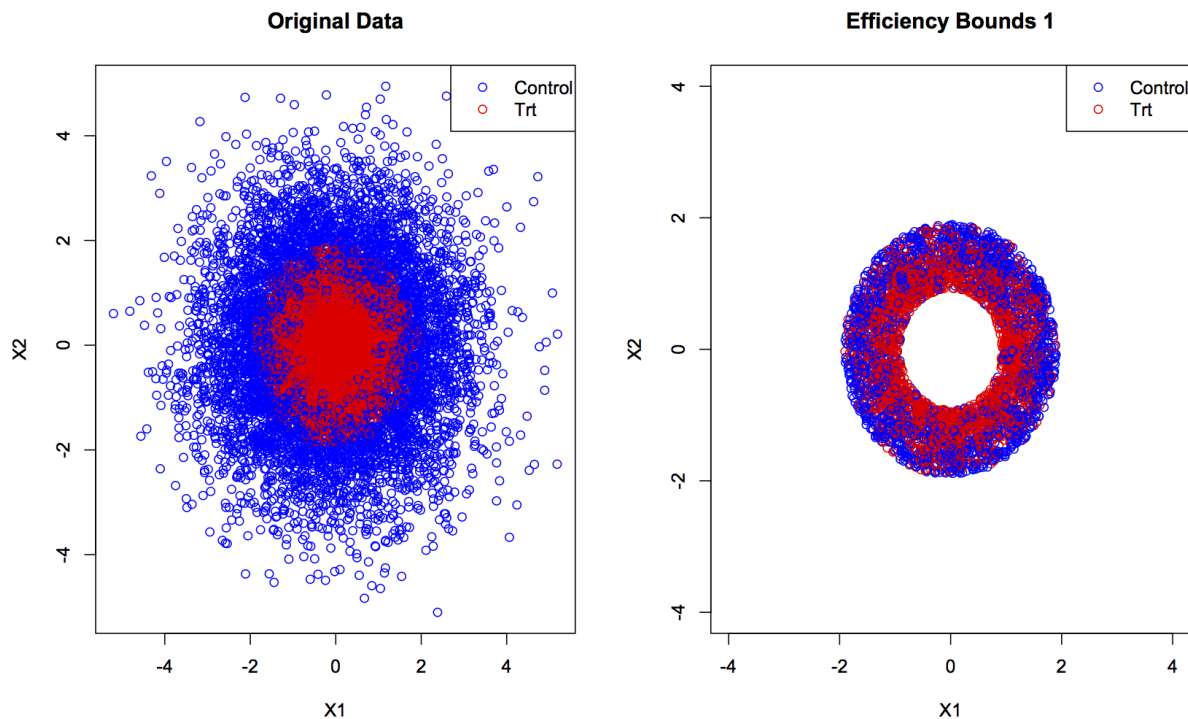


Figure 3.1: *Common support region Of Efficiency Bounds on Real Propensity Score*

using propensity match. The estimations are very close to the true treatment effect of 2 for homogeneous model. For the heterogeneous model, the estimated ATT is 1.91 using genMatch and 1.9078 using propensity match. The estimations are very close to the true treatment effect of 1.906682 for heterogeneous model. And the true ATT of common support is close to the true ATT of sample data of 1.945225. Because we can hardly know propensity scores in real world, we will next test this method using the well estimated propensity scores of

$logit[\hat{e}(\mathbf{x}_i)] = \beta_1(x_{i1} + x_{i2} + x_{i3})^2 + \beta_2x_{i1}^2 + \beta_3x_{i2}^2 + \beta_4x_{i3}^2$. We will also cut the experiment units using $[10\%,90\%]$ boundaries. Figure 3.2 is the graph of x_1 versus x_2 of common support region and the original data set.

Figure 3.2 is the graph for both homogeneous model and heterogeneous model. From Figure 3.2, we see that, for this example, Efficiency Bounds [Crump et al. \(2009\)](#) gives a good common support when estimation of propensity scores is good. Using the well estimated propensity scores, for homogeneous model we get the estimated ATTs of 1.998 by genMatch

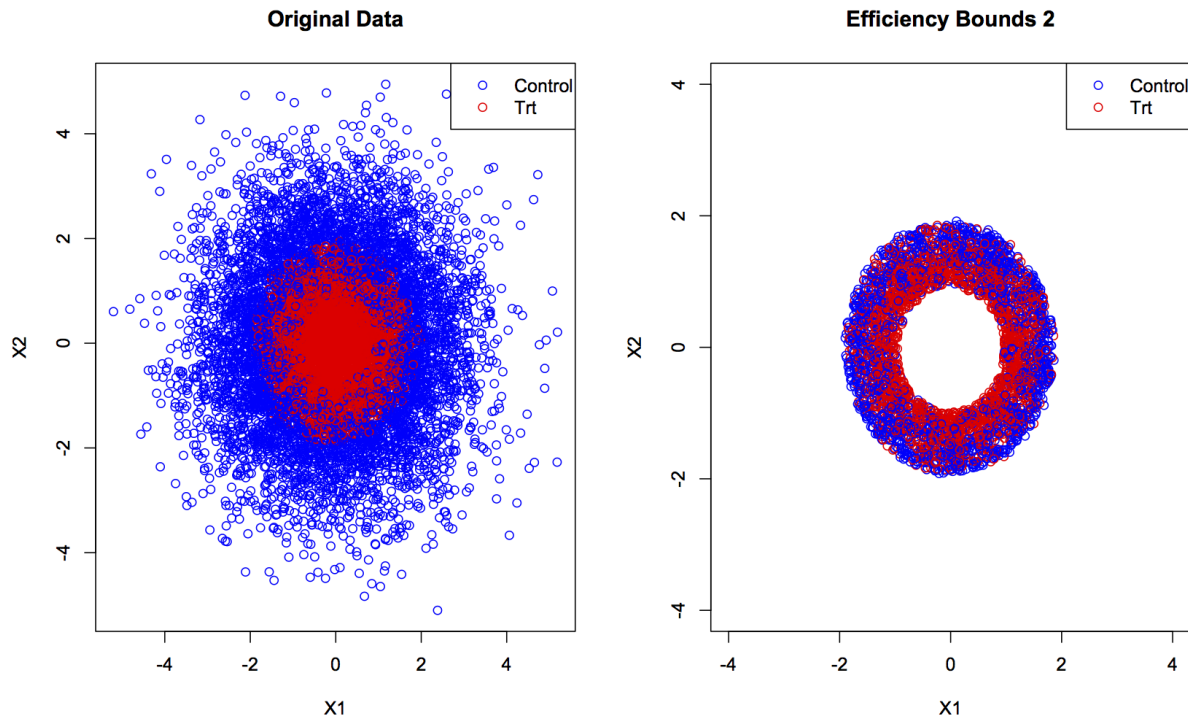


Figure 3.2: *Common support region Efficiency Bounds on Estimated Propensity Score*

Efficiency Bounds(Estimated Propensity Score)		
<i>Common Support</i>		
Running Time	Common Support Size	# Treated Units
Instant	3771	2040
<i>Estimated ATT</i>		
	GenMatch	Propensity Match
Estimated ATT	1.998	1.9964
AI-SE	0.0025599	0.0086073
P-Value	<2.22e-16	<2.22e-16

Table 3.2: *results of efficiency bounds using well estimated propensity score*

and 1.9964 by one to one propensity match. They are very close to true treatment effect of 2. For heterogeneous model we get the estimated ATTs of 1.9367 by genMatch and 1.9356 by propensity match. They are very close to the true ATT of heterogeneous common support of 1.939173. And the true ATT of common support is close to the true ATT of sample data of 1.945225. Efficiency Bounds [Crump et al. \(2009\)](#) works well if we have a good estimation of propensity scores. The question is do we always have a good estimation of propensity

scores or what would happen if we don't have well estimated propensity scores. Now suppose we know nothing about the true propensity scores and we estimate propensity scores with logistic linear model:

$$\text{logit}(\hat{e}(x)) = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$$

let us also cut on range [10%, 90%] for these badly estimated propensity scores to get a common support. Figure 3.3 is the graph of x_1 versus x_2 comparing common support region with sample data set for both homogeneous model and heterogeneous model.

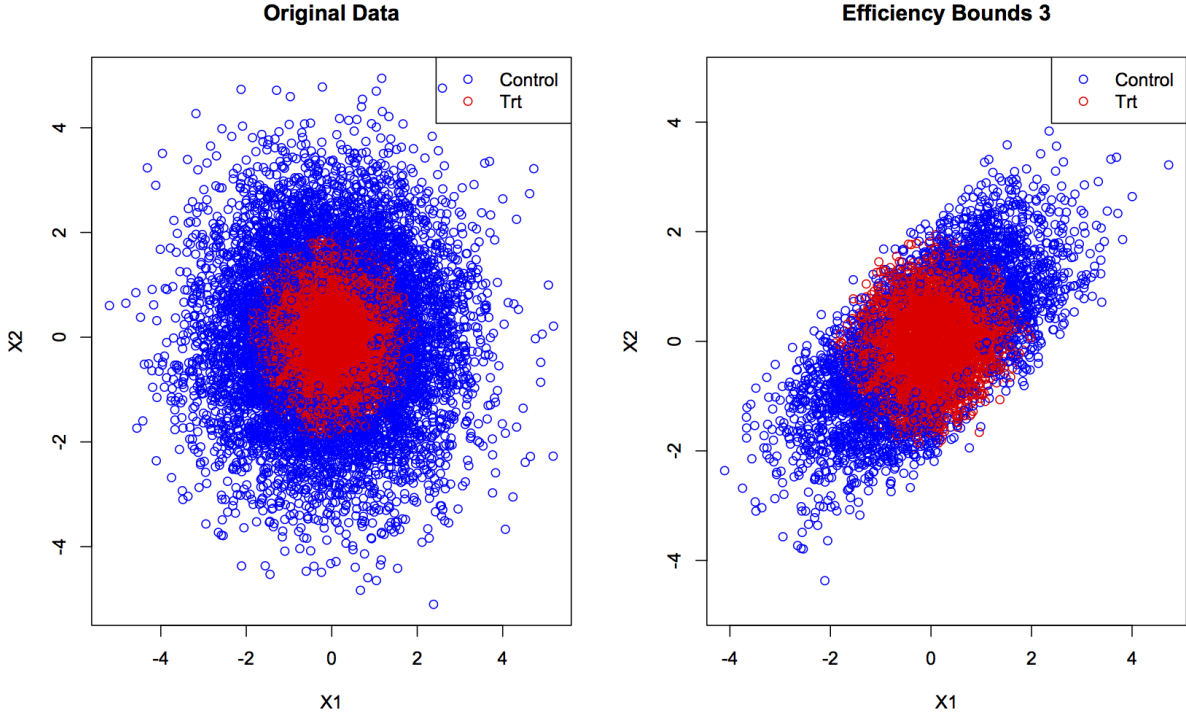


Figure 3.3: *Common support region Efficiency Bounds on Badly Estimated Propesnisty Score*

From figure 3.3, we see the common support is obviously biased. This common support region favors the experiment units with covariate x_1 close to covariate x_2 . The distribution of propensity scores is still very dispersed. For homogeneous model, the result of estimated ATTs from the common support region are 4.8336 using genMatch and 5.0322 using one to one propensity match. They are twice larger than the true treatment effect of 2 for homogeneous model. Table 3.3 shows the time to generate common support and result

for homogeneous model. For heterogeneous model, the result of estimated ATTs from the

Efficiency Bounds(Badly Estimated Propensity Score)		
<i>Common Support</i>		
Running Time	Common Support Size	# Treated Units
Instant	6000	3550
<i>Estimated ATT</i>		
	GenMatch	Propensity Match
Estimated ATT	4.8336	5.0322
AI-SE	5.0287	3.2
P-Value	0.33645	0.11582

Table 3.3: results of efficiency bounds using badly estimated propensity score

common support region are 4.7838 using genMatch and 4.5084 using one to one propensity match. They are very different from the true heterogeneous model ATT of common support of 1.966739.

In summary, Efficiency Bounds [Crump et al. \(2009\)](#) is highly restricted and its performance depends solely on the how well the propensity scores are estimated.

3.4 Maxbox Method

Discrete Optimization for Interpretable Study Populations and Randomization Inference in an Observational Study of Severe Sepsis Mortality ([Fogarty et al., 2016](#)) gives the Maxbox method. A maxbox functions is also provided by the author ([Fogarty et al., 2016](#)). A maxbox can be written as $XL \ll X_i \ll XU$. It cuts all the experiment units with $X_i \ll XL$ or $X_i \gg XU$ while includes as many experiment units as possible. Let X_i be the vector of covariates for experiment unit i, and $X_i = (x_{i1}, x_{i2}, x_{i3})$. The maxbox boundaries are $XL = (XL_1, XL_2, XL_3)$ for lower bounds and $XU = (XU_1, XU_2, XU_3)$ for upper bounds. For each experiment unit i in the common support region, it satisfies that $XL_j < X_{ij}$ and $X_{ij} < XU_j$ for $j=1,2,3$. The maxbox always generates a convex rectangular region. Now lets check the performance of Maxbox [Fogarty et al. \(2016\)](#). Suppose we should exclude all units with propensity scores either smaller than 0.1 or larger than 0.9. Then let's use the built in maxbox function [Fogarty et al. \(2016\)](#) to get the common support. My computer

has a processor of 1.6 GHz Intel Core i5 and a memory of 8 GB 1600 MHz DDR3.

First lets use the true propensity scores to find the common support. Figure 3.4 is the common support of x_1 versus x_2 for both homogeneous model and heterogeneous model.

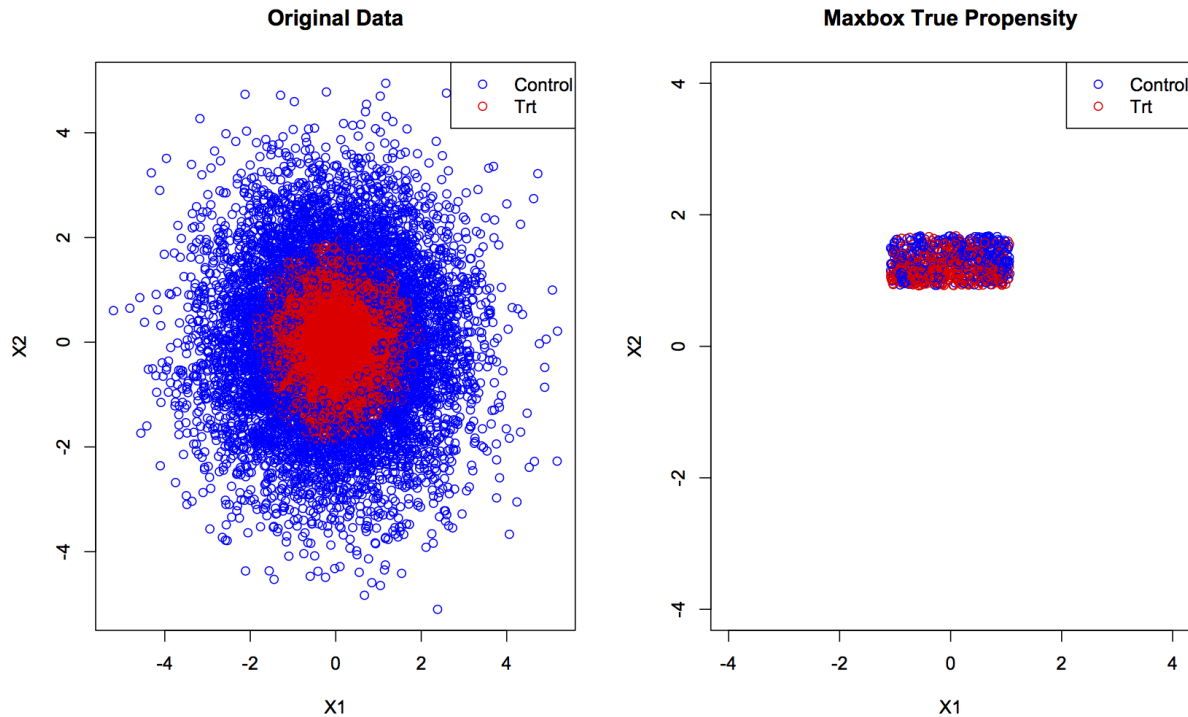


Figure 3.4: *Common support region Maxbox on True Propensity*

Figure 3.4 is the common support for both homogeneous model and heterogeneous model. From Figure 3.4 we see that the common support region generated by Maxbox (Fogarty et al., 2016) is very small, because it is a convex region cut out from a non-convex region. From total 10000 experiment units, we have only 803 experiment units including 441 treated units in the common support. Less than 10% of the experiment units survived the maxbox algorithm. It is not a very efficient way to use experiment units. Further when the common support region is too small, the variance of the estimated ATT tends to be large and the danger of having a large bias will be significant. The algorithm time for Maxbox (Fogarty et al., 2016) is more than 40 seconds on my computer. It takes too much time using propensity scores compared with LCC method. Table 3.4 contains the time to generate time for both homogeneous model and heterogeneous model and the result for homogeneous model. For the homogeneous

Maxbox(True Propensity Score)		
<i>Common Support</i>		
Running Time	Common Support Size	# Treated Units
43.03392 secs	803	441
<i>Estimated ATT</i>		
	GenMatch	Propensity Match
Estimated ATT	2.0005	2.0084
AI-SE	0.0049227	0.018291
P-Value	<2.22e-16	<2.22e-16

Table 3.4: *results of maxbox using true propensity score*

model, although the estimated ATTs are not too bad, the standard errors of estimated ATTs using genMatch and one to one propensity match are 0.0049227 and 0.018291. Compare them with those from efficiency bounds (Crump et al., 2009), the standard errors of estimated ATT using Maxbox Fogarty et al. (2016) are larger. For the heterogeneous model, the true ATT for the common support is 5.588091 which is very different from the true ATT of data sample of 1.945225. The estimated ATT using genMatch is 5.5701 and it is 5.5745 using propensity match. They are close to the true ATT of the common support.

Figure 3.5 is the result using well estimated propensity scores for both homogeneous model and heterogeneous model, and the results are similar to those using true propensity scores. The time to generate common support and the results of estimated ATT using genMatch and one to one propensity match for homogeneous model is in table 3.5

Maxbox(Estimated Propensity Score)		
<i>Common Support</i>		
Running Time	Common Support Size	# Treated Units
42.15375 secs	677	392
<i>Estimated ATT</i>		
	GenMatch	Propensity Match
Estimated ATT	1.986	1.9874
AI-SE	0.0059696	0.018989
P-Value	<2.22e-16	<2.22e-16

Table 3.5: *results of maxbox using well estimated propensity score*

For heterogeneous model, the true ATT of the common support is -1.851055 which is very different from the true ATT of sample data of 1.945225. The estimated ATT using

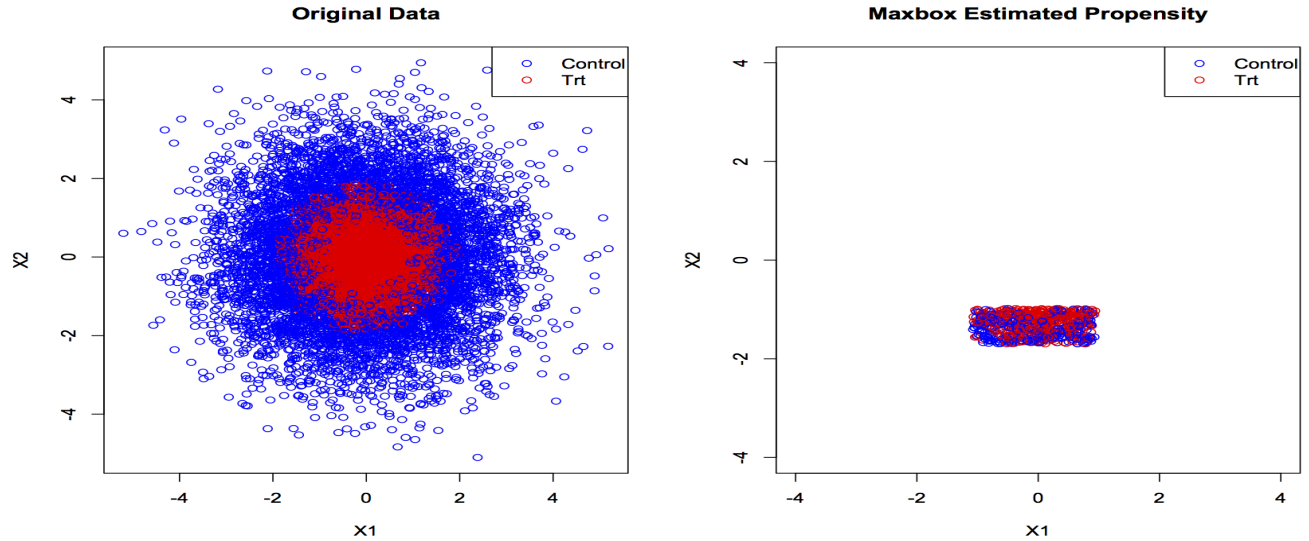


Figure 3.5: *Common Support Maxbox on Estimated Propensity*

GenMatch is -1.8654 using genMatch and -1.8637 using propensity match.

Generally speaking, Maxbox [Fogarty et al. \(2016\)](#) only generates convex common support region. It can hardly generate a common support region with sufficient size for a non-convex sample data set. This makes Maxbox [Fogarty et al. \(2016\)](#) a very inefficient way to use experiment units. For the well estimated propensity scores we have only 677 experiment units in common support. The consequence is that the variance of the estimated ATT becomes larger. Second Maxbox [Fogarty et al. \(2016\)](#) takes long algorithm time. In our experiment, for both true propensity score and well estimated propensity scores, the algorithm time exceeds 40 seconds. Further, for heterogeneous model, the true ATT of common support is very different from the true ATT of sample data, which means the common support does not represent the sample well.

3.5 BART

The idea of BART ([Hill and Su, 2013](#)) is very straight forward. Let $Y = f(z, x) + \varepsilon$, where $\varepsilon \sim N(0, \sigma^2)$ and $f(z, x) = g(z, x; T_1, M_1) + g(z, x; T_2, M_2) + \dots + g(z, x; T_m, M_m)$. Let each (T_j, M_j) denote a single subtree model. Then each iteration of the BART Markov Chain

generates a new draw of f from the posterior distribution. Let f^r denote the r th draw of f . Then compute $d_i^r = f^r(1, x_i) - f^r(0, x_i)$, for $i=1, 2, \dots, n$. Let $s_i^{f_0}$ and $s_i^{f_1}$ be the standard deviations of the draws of $f(0, x_i)$ and $f(1, x_i)$ respectively for the i th observation.

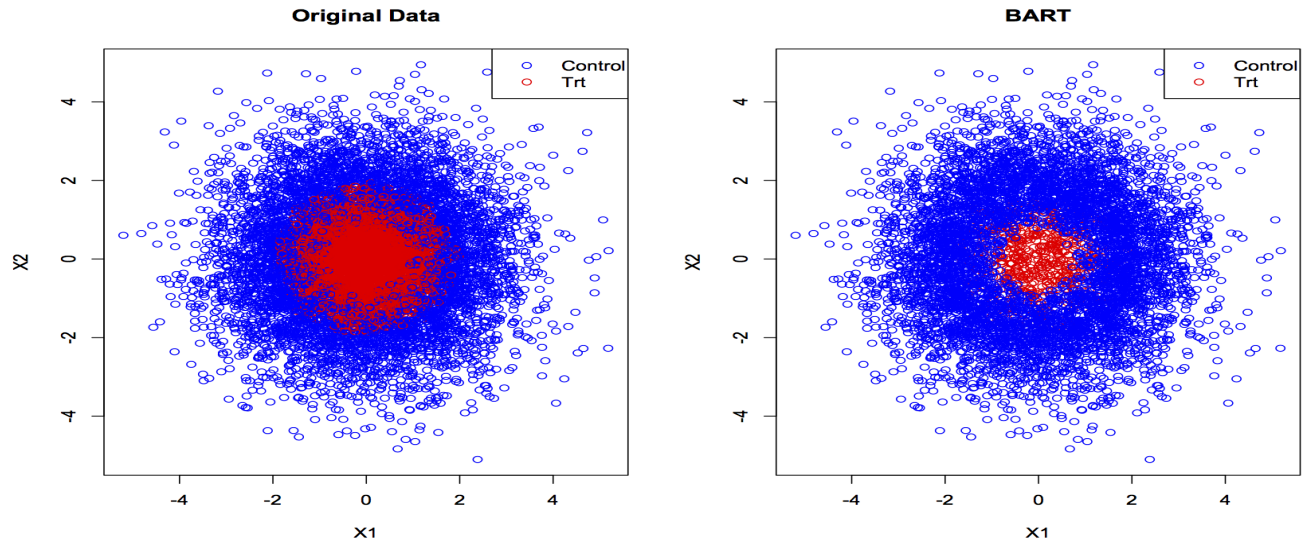


Figure 3.6: *Common support Bart Homogeneous Model*

Then for $z_i = a$, where $a=0$ or 1 , we cut off the units such that $s_i^{f_{1-a}}/s_i^{f_a} > 2.706$ for $\alpha = 0.1$ significance level. Bart directly matches the data and gives the ATT. Figure 3.6 is the graph of x_1 versus x_2 comparing common support region and sample data set for homogeneous model. Figure 3.7 is the graph of x_1 versus x_2 comparing common support region and sample data set for heterogeneous model. For homogeneous model, the results of estimation on ATT are in table 3.6. The BART gives the estimated ATT of 2.795, which is not very precise. Working on the common support, the estimated ATT using GenMatch is 2.4866 and is 2.3645 using one to one propensity match. For heterogeneous model, the true ATT of common support is 1.945225. The Bart gives the estimation of ATT of 3.585, which is not very precise.

For BART (Hill and Su, 2013), instead of using propensity scores, we can use observable covariates X_1, X_2, X_3 . This is very useful, especially when there is no good estimation of the propensity scores. The algorithm takes about 25 seconds which is good. However, BART (Hill and Su, 2013) gets only 589 treated units for homogeneous model. This may

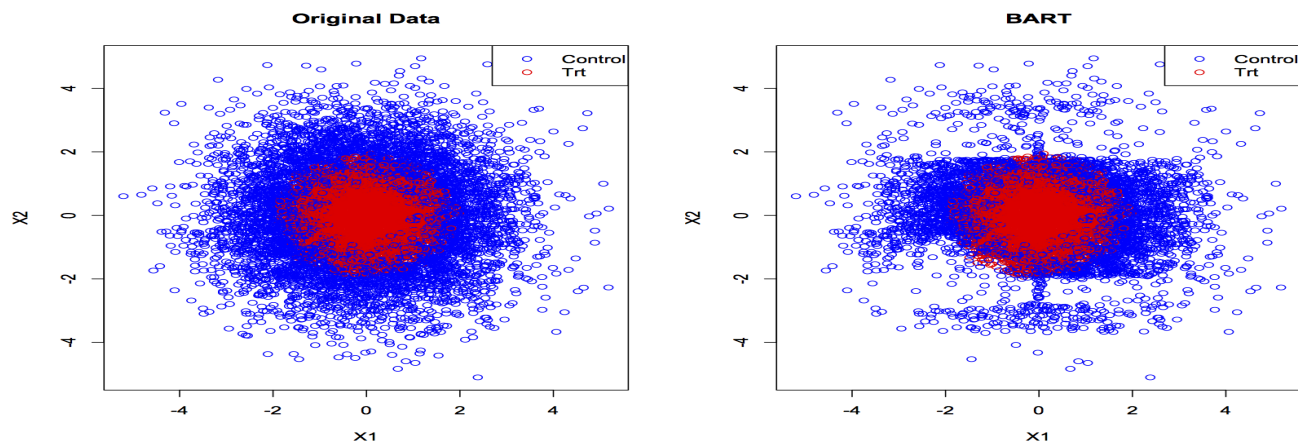


Figure 3.7: Common support Bart Heterogeneous Model

Bart(alpha=0.10)		
<i>Common Support</i>		
Running Time	Common Support Size	# Treated Units
25.66926 secs	6477	589
<i>Estimated ATT</i>		
	GenMatch	Propensity Match
Estimated ATT	2.4866	2.3645
AI-SE	0.24938	0.89732
P-Value	<2.22e-16	0.008412

Table 3.6: results of bart for $\alpha = 0.1$ significance level

leads to some bias for estimation of ATT and also may increase the variance of the estimated ATT. For homogeneous model, using genMatch and propensity match, the estimated ATT are 2.4866 and 2.3645 respectively, which are close to the true treatment effect but not very precise. The standard errors of ATTs from propensity match is 0.008412, which are significantly larger than those using efficiency bounds and maxbox. Bart also takes 4.017369 minutes algorithm time which is not very fast. For our sample data, BART (Hill and Su, 2013) might not be a very good solution to get common support.

3.6 Optimal Match

The idea of Optimal Match (Rosenbaum, 2012) is simple. In Optimal Match, each experiment unit received treatment is paired with an experiment unit received control. The treated units which can not be paired are removed. The data should be paired such that the average distance within each pair is minimized. The type of distance could be chosen for researcher's interest. It could be general euclidean, mahalanobis or the difference between propensity scores. The good part of Optimal Match (Rosenbaum, 2012) is that we can use both the observable variables and the propensity scores to find the common support region.

An r package called rcbalance (Rosenbaum, 2012) is developed for optimal match. It creates the mahalanobis distance between every control unit and every treated unit. The return value is a matrix of two columns. The first column is the index of the treated units, and the second column is the index of control units paired with the corresponding treated units. A good and unique part for rcbalance is that, the results are already paired. We don't need to do genMatch or 1 to 1 propensity match. We can directly calculated the ATT on those paired units.

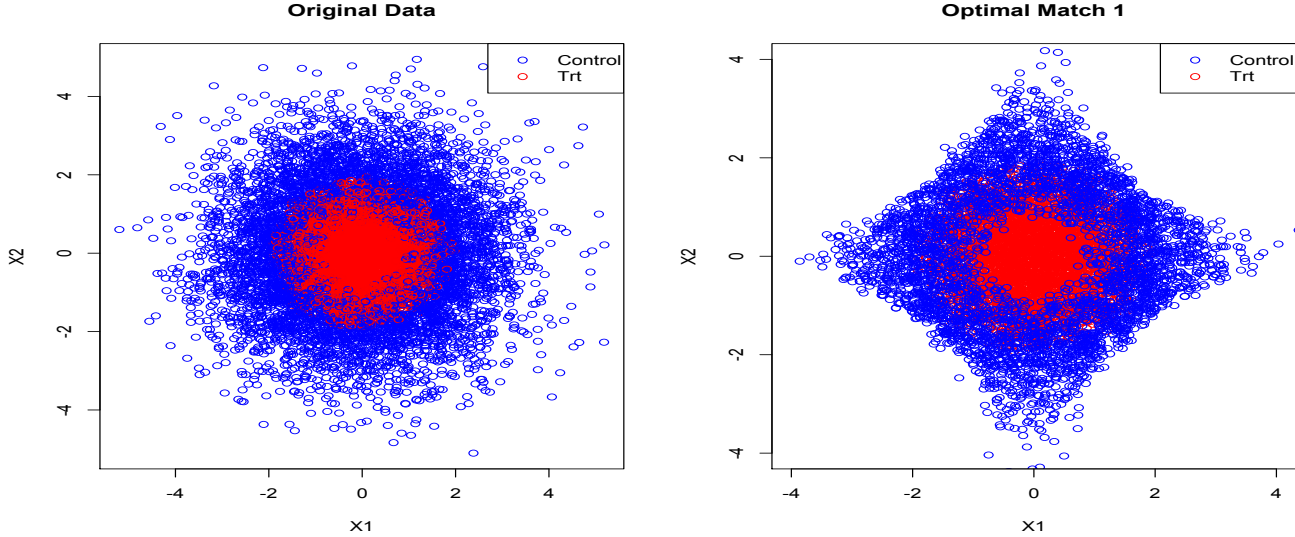


Figure 3.8: Common support Optimal Match

Figure 3.8 is the common support compared with the sample data set for both homo-

geneous model and heterogeneous model. This graph is diamond shaped. We don't need to worry too much about the shape of common support and propensity balance, because the units are already paired. Now let's calculate the ATT using the matched pairs. For homogeneous model, if we take the average of the treatment effects of all pairs, then the ATT is 4.629711, which is not precise. Now let's take the median of the treatment effects of all pairs, then ATT is 2.237223, which is close to true treatment effect of 2. If we cut the matched pairs with smallest 500 treatment effects and largest 500 treatment effect, then ATT is 2.174696, which is close but still not close enough. For heterogeneous model, the true ATT of common support is 1.945225. Similarly, let us first calculate the difference of responses for each pair for heterogeneous model. Using the differences of all pairs from common support, the estimated ATT is 4.574936, which is far from the true ATT of common support. If we cut the lowest 500 differences, and highest 500 differences, the estimated ATT is 2.542459. If we use the median of the differences, the estimated ATT is 2.53921, which is better but still far from the true ATT of common support. The run time for optimal match is 10.6284 hours.

Generally speaking, optimal match can take advantages of observable covariates and can offer a somehow close estimation. It also gives the paired matches, which is good. However, optimal match takes huge amount of time and comparable to LCC, the result is still worse.

3.7 Convex Hull

The idea of Convex Hull (King and Zeng, 2006) is to first get a convex region for the treated unit, and include all the control unit in this convex region in the common support. Then get a convex region for the control unit and include all the control unit in this convex region in the common support. (King and Zeng, 2006) offers the whatif functions. This functions offers two distance types, the gower distance and the euclidean distance. The gower distance is defined as $G_{ij} = \frac{1}{k} \sum_{i=1}^k \frac{|x_{ik} - x_{jk}|}{r_k}$, where $r_k = \max(X_{.k}) - \min(X_{.k})$ King and Zeng (2006).

Let's first use gower distance and euclidean distance on covariates to generate the common support. For our sample data, using gower distance and euclidean distance on covariates will

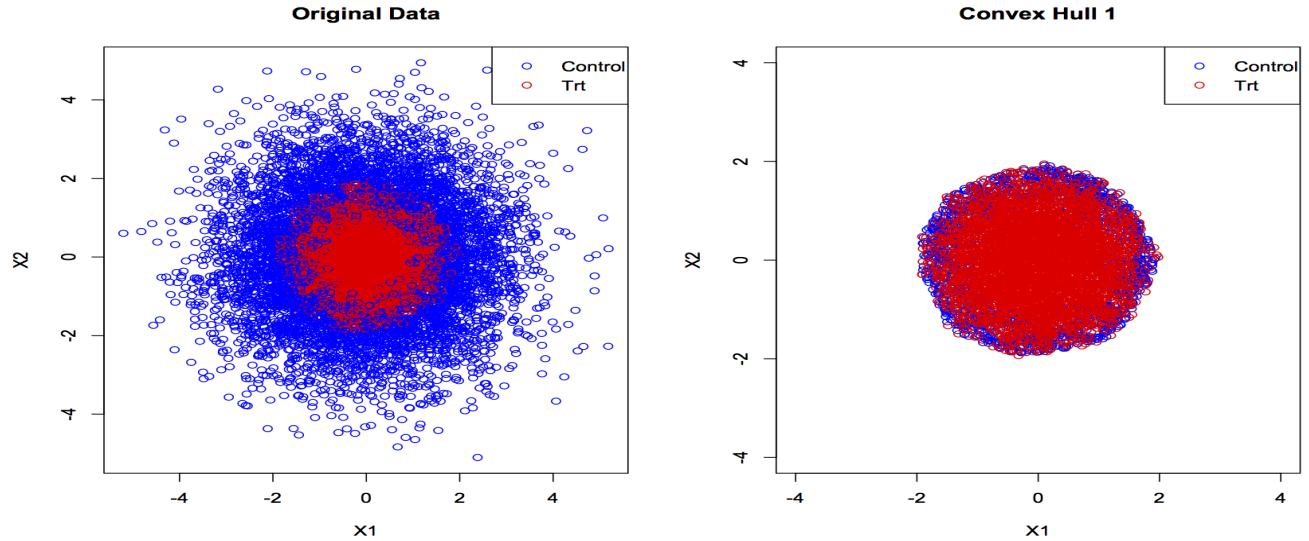


Figure 3.9: *Common support Convex Hull on Covariates*

give same results. Figure 3.9 is the result of X_1 vs X_2 using covariates comparing common support and data sample for both homogeneous model and heterogeneous model. Because the treated units are gathered at the center, only a few of the control units were included in the convex hull of the treated units. Another thing is that, the algorithm ignores the hole in the control units, and only used its outer bound to create the convex hull. We have totally 5091 units including 4109 treated units in the common support. For homogeneous model, the result for estimated ATT using GenMatch is 1.0122 and using one to one propensity match is 1.0307. It is a bad estimation. For heterogeneous model, the true ATT for common support is 1.945135. The estimated ATT is 4.3045 using genMatch and 4.129 using propensity match. The estimation of ATT for heterogeneous model is bad. The algorithm time is 1.32166 minutes gower distance and 1.18139 minutes for euclidean distance.

One solution is to decrease the dimension. In extreme case, a convex hull for one dimension is always available. Now let's use the well estimated propensity scores as one dimension measure for distance. Figure 3.10, we see the graph of the common support is not convex. Because, we use the convex hull on estimated propensity scores, and convexity of the propensity score is not the convexity of covariates. The common support has treated and control units fairly closely distributed. This algorithm takes 42.11525 seconds using euclidean dis-

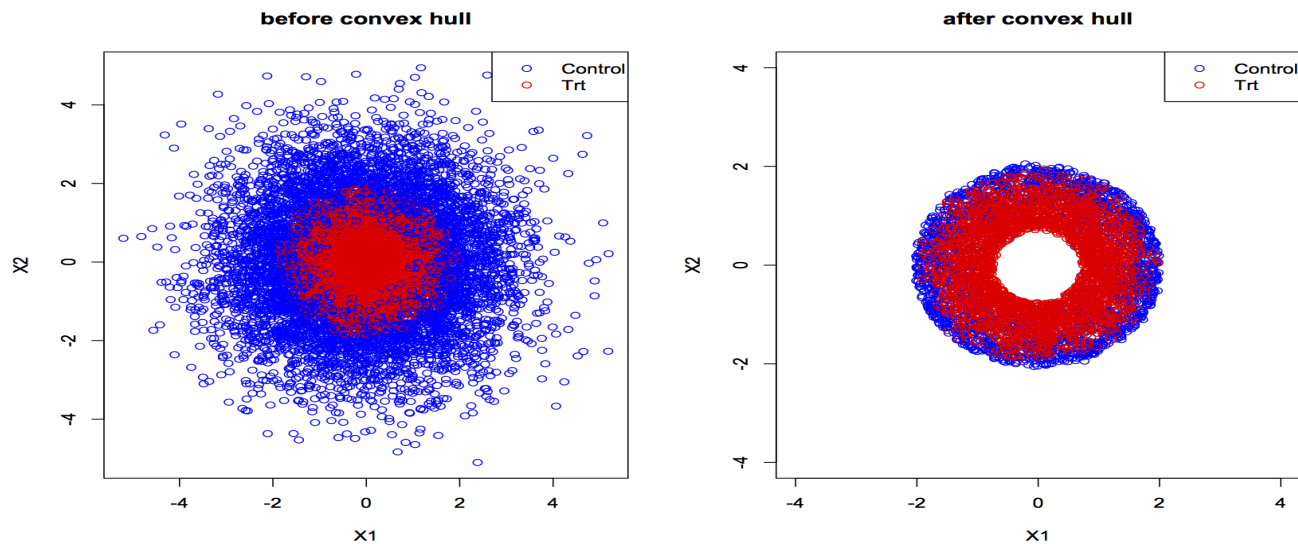


Figure 3.10: *Common support Convex Hull on Estimated Propensity*

tance and 49.25104 seconds for gower distance. For homogeneous model, the estimated ATT is 2.0162 using genMatch and 1.9971 using propensity match, which are very close to the true ATT of 2. For heterogeneous model, the estimated ATT is 1.9435 using genMatch and 1.927574 using propensity match. They are close to the true ATT of common support of 1.927574 and true ATT of sample data of 1.945225.

Generally speaking, Convex Hull (King and Zeng, 2006) has its limitation. It highly requires the shape of your data, or otherwise you must have a good estimation of propensity scores.

3.8 LCC Methods

Now let's use LCC methods to get the common support. Compared to the above methods, LCC has the following advantages. First it offers great flexibility. It can use either observable covariates or propensity scores to generate the common support. The distance could be mahalanobis, euclidean and largest caliper. Second LCC works on both convex and non-convex regions and can generate both convex and non-convex common support regions. Third the algorithm time of LCC is faster than any of the previous methods. The algorithm

time is $O(n^2)$ and memory space used is $O(n)$. The R package `llc` provides the algorithm for LCC methods. It could be founded in my URL.

For our sample data choosing threshold that cuts off 40% units would be a good start. We will keep this. First let's run LCC on true propensity scores. Figure 3.11 is the graph of X_1 vs X_2 with true propensity scores for both homogeneous model and heterogeneous model. From Figure 3.11, after LCC the distribution of the the data points are getting closer, and there is a hole in the common support. The absence of treated units in the center has been taken care by LCC, so there are no treated units in the center of common support, because we have no close counterfactual control units for treated units at center. For homogeneous model, the estimated ATT on common support using `GenMatch` is 2.0074 and using one to one propensity match is 2.0095. It is a good estimation. For heterogeneous model, the true ATT of common support is 1.897989. The estimated ATT is 1.9061 using `genMatch` and 1.9026 using propensity match. They are close to the true ATT of sample data of 1.945225. The running time is 10.44532 seconds, which is short for 10000 data points.

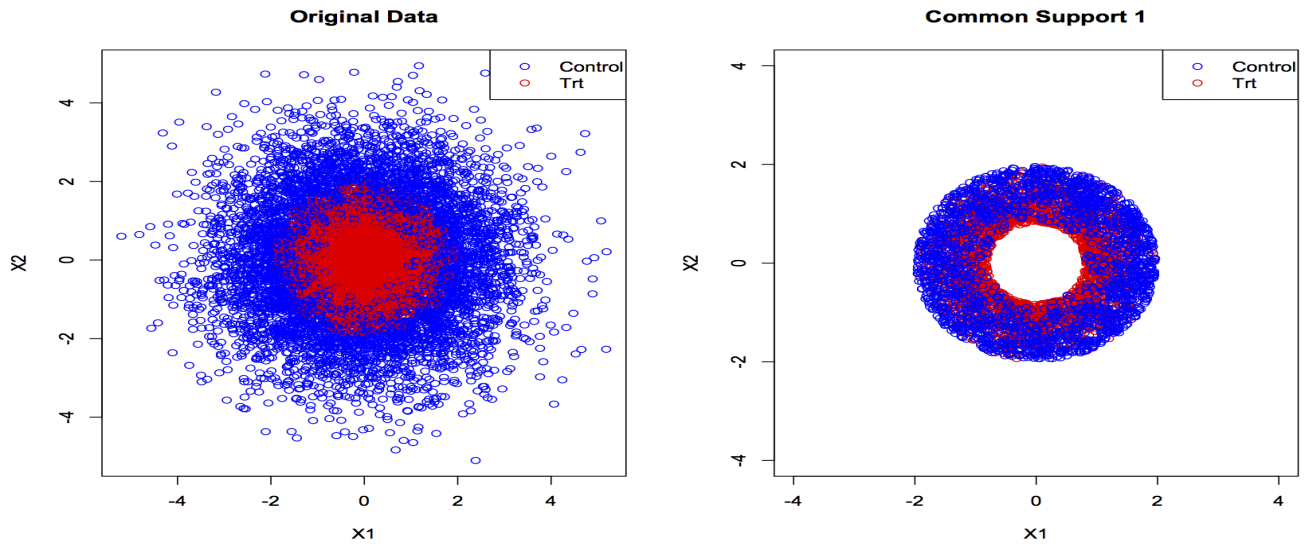


Figure 3.11: *Common support LCC on True Propensity*

Second let's run LCC on our estimated propensity scores. Figure 3.12 is the data distribution for X_1 vs X_2 comparing common support and the sample data. The lack of counterfactual for treated units at the center has been take care by LCC, so the common support has

no points at the center. For homogeneous model, using this common support, the estimated ATT is 2.0166 using GenMatch and 1.9857 using one to one propensity match. They are good estimations and close to the true effect of 2. For heterogeneous model, the true ATT is 1.947691. The estimated ATT is 1.9716 using genMatch and 1.9334 using propensity match. They are close to the true ATT of sample data of 1.945225. The running time is 9.966149 seconds, which is excellent.

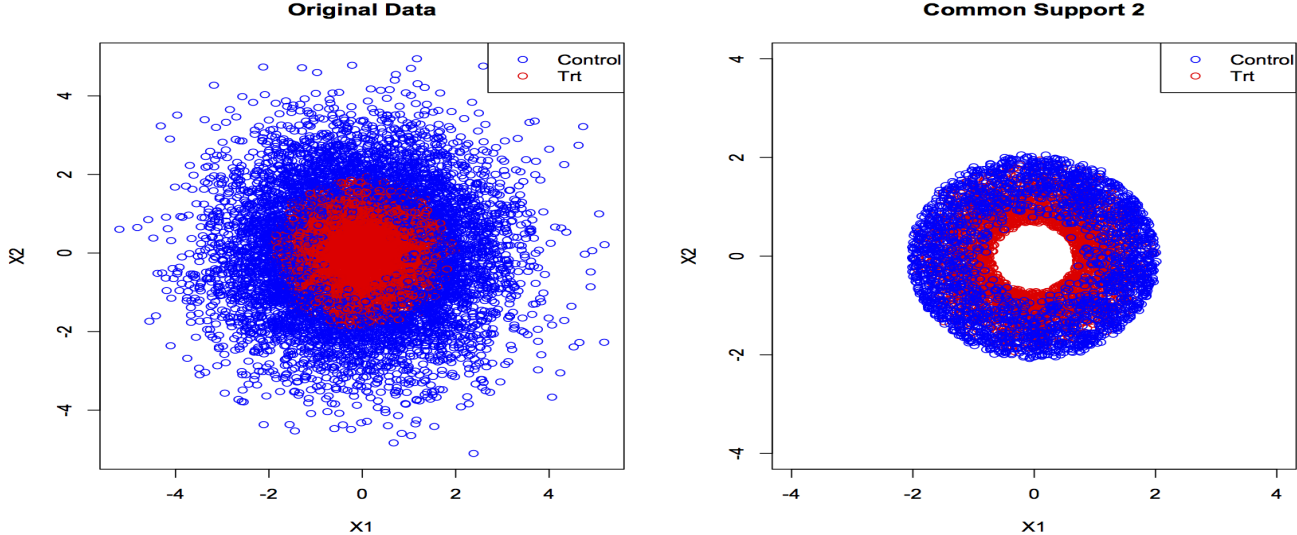


Figure 3.12: *Common support LCC on Estimated Propensity*

Now let’s suppose we don’t have a good estimation of propensity scores, and let’s use the observable covariates. First Let’s try general euclidean. Figure 3.13 is the data distribution for X_1 vs X_2 comparing common support with sample data for both homogeneous model and heterogeneous model. The lack of counter factual for treated units at the center has been taken care by LCC. For homogeneous model, with this common support, the estimated ATT using GenMatch is 2.1041 and using one to one propensity match is 2.0507, which are close to the true effect of 2. For heterogeneous model, the true ATT of common support is 1.928466. The estimated ATT is 2.0128 using genMatch and 1.9495 using propensity match. They are close to the true ATT of sample data of 1.945225. The algorithm time is 33.53909 seconds. Using general euclidean on covariates for our sample data, LCC works well.

Now let’s try mahalanobis distance. Figure Figure 3.14 is the data distribution for X_1

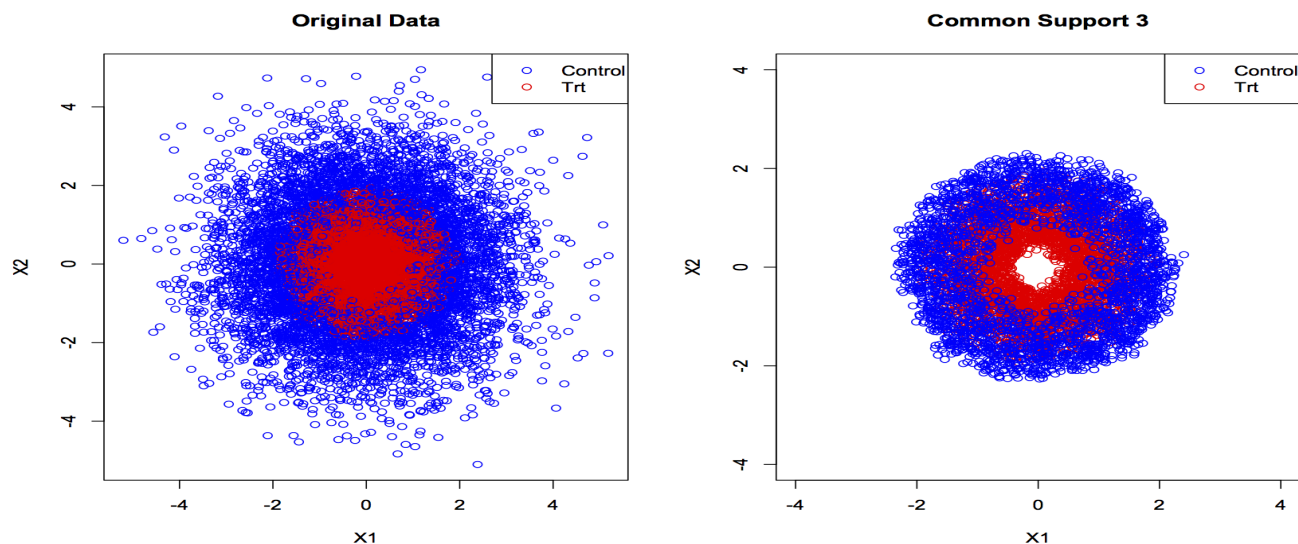


Figure 3.13: *Common support LCC General Euclidean*

vs X_2 for both homogeneous model and heterogeneous model. The lack of counterfactual for treated units at the center has been taken care by LCC, and the common support has no points at the center. For homogeneous model with this common support, the estimated ATT is 2.1848 using GenMatch and it is 2.1056 using one to one propensity match. They are a little bit worse than those using general euclidean distance but still good enough. For heterogeneous model, the true ATT of common support is 1.937921. The estimated ATT is 2.1232 using genMatch and 2.0137 using propensity match. They are close to the true ATT of sample data of 1.945225. The algorithm time is 7.012911 minutes, which is longer than that using general euclidean distance. This is because mahalanobis distance is more complicated than general euclidean distance. For now LCC is the still the fastest algorithm using mahalanobis distance and its common support still gives the most accurate estimation for ATT using mahalanobis distance.

Now let's use the Largest Caliper distance next. Figure 3.15 is the data distribution for X_1 vs X_2 for both homogeneous model and heterogeneous model. The lack of counterfactual for treated units at the center has been taken care by LCC and the common support has no points at the center. For homogeneous model with common support after LCC using Largest Caliper, the estimated ATT is 2.0359 using GenMatch and it is 2.0354 using one

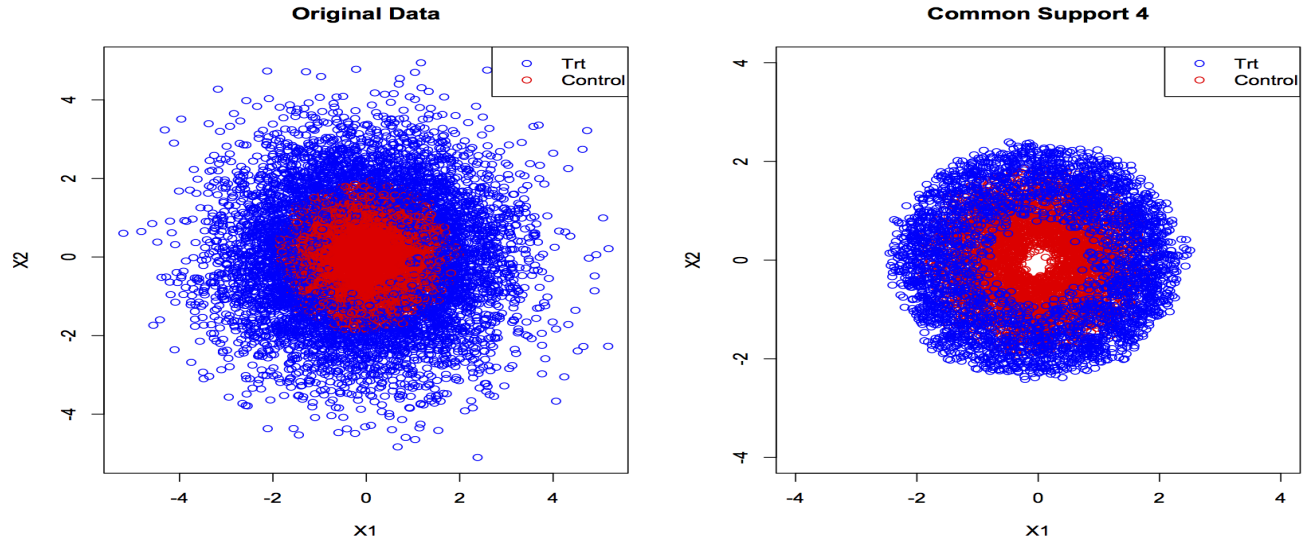


Figure 3.14: *Common support LCC Mahalanobis Distance*

to one propensity scores match. The estimation is very close to the true effect of 2. For heterogeneous model, the true ATT of common support is 1.94954. The estimated ATT is 1.9847 using genMatch and 1.9569 using propensity match. They are close to the true ATT of sample data of 1.945225. The algorithm time is 59.08531 seconds, which is good. So LCC gives a good common support for our sample data using largest caliper.

Now we have shown that, LCC can generate a good common support using propensity scores and using observable covariates with different distance measures for both homogeneous model and heterogeneous model. Another thing to notice to notice is that. In original data, we have 4112 treated units and 5882 control units. The common support using true propensity scores has 2541 treated units and 2080 control units. Using estimated propensity scores, there are 2927 treated units and 2369 control units. Using general euclidean distance on covariates, there are 3378 treated units and 2844 control units. Using Mahalanobis distance, there are 3724 treated units and 3261 control units. Using Largest Caliper, there are 2867 treated units and 2309 control units. With all measures used by LCC to generate common support, the number of treated units and control units is well balanced.

Let's compare the LCC with other methods. Using propensity scores, compared with Maxbox (Fogarty et al., 2016), LCC offers a larger and less biased common support region

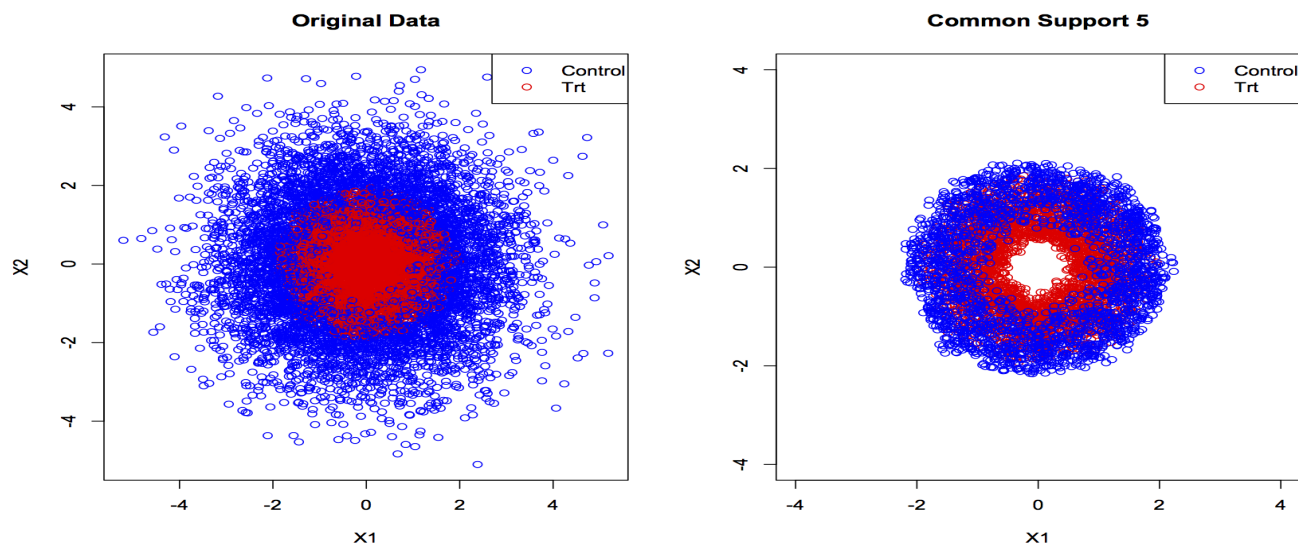


Figure 3.15: *Common support LCC Largest Caliper*

and takes approximately one fourth algorithm time used by Maxbox (Fogarty et al., 2016). Also the common support of Maxbox fails to represent the sample data because the true ATT of common support is very different to the true ATT of sample data. Compared with Efficiency Bounds (Crump et al., 2009), LCC offers more flexibility. Both Maxbox (Fogarty et al., 2016) and Efficiency Bounds (Crump et al., 2009) are highly relied on estimation of propensity scores, while LCC can work on other distance measures and taking advantages of the observable variables. Compared with Optimal Match (Rosenbaum, 2012), LCC takes a much shorter algorithm time and more precise results. Using general euclidean and largest caliper, LCC algorithm times are less than 1 minutes. With Mahalanobis distance the algorithm time is 7 min. Using estimated Propensity scores, the algorithm time is 9.996 seconds. Using true propensity score, the algorithm time is 10.4453 seconds. Comparing with BART (Hill and Su, 2013), the common support of LCC has more data points and offers a more precise estimation. BART only works on observable covariates, and its estimated ATT is around 2.4 is not very precise for both homogeneous model and heterogeneous model.

LCC(True Propensity Score)		
<i>Common Support</i>		
Running Time	Common Support Size	# Treated Units
10.44532 secs	4626	2541
<i>Estimated ATT</i>		
	GenMatch	Propensity Match
Estimated ATT	2.0074	2.0095
AI-SE	0.0040435	0.012223
P-Value	<2.22e-16	<2.22e-16

Table 3.7: results of LCC using true propensity score

LCC(Estimated Propensity Score)		
<i>Common Support</i>		
Running Time	Common Support Size	# Treated Units
9.966149 secs	5296	2927
<i>Estimated ATT</i>		
	GenMatch	Propensity Match
Estimated ATT	2.0166	1.9857
AI-SE	0.0074541	0.022081
P-Value	<2.22e-16	<2.22e-16

Table 3.8: results of LCC using well estimated estimated propensity score

LCC(General Euclidean)		
<i>Common Support</i>		
Running Time	Common Support Size	# Treated Units
33.53909 secs	6222	3378
<i>Estimated ATT</i>		
	GenMatch	Propensity Match
Estimated ATT	2.1041	2.0507
AI-SE	0.034776	0.072227
P-Value	<2.22e-16	<2.22e-16

Table 3.9: results of LCC using general euclidean distance

LCC(Mahalanobis)		
<i>Common Support</i>		
Running Time	Common Support Size	# Treated Units
7.012911 mins	6985	3724
<i>Estimated ATT</i>		
	GenMatch	Propensity Match
Estimated ATT	2.1848	2.1056
AI-SE	0.042975	0.14635
P-Value	<2.22e-16	<2.22e-16

Table 3.10: *results of LCC using mahalanobis distance*

LCC(Largest Caliper)		
<i>Common Support</i>		
Running Time	Common Support Size	# Treated Units
59.08531 secs	5176	2867
<i>Estimated ATT</i>		
	GenMatch	Propensity Match
Estimated ATT	2.0359	2.0354
AI-SE	0.015386	0.034242
P-Value	<2.22e-16	<2.22e-16

Table 3.11: *results of LCC using the largest caliper*

3.9 Discussion

In the future research. There are still many thing can be studied. First, how to select the threshold could be still studied. For now I set the threshold such that approximately 40% of the experiment units will be excluded from the common support. There could be some better selection for the threshold. Second there could be more measures on distances. For our sample, the running time using general euclidean is 33 seconds. It is 7 minutes using mahalanobis distance. The latter is is approximately 14 times of the previous. Also the estimated ATT using common support by LCC with mahalanobis distance is less precise. We can not say that Mahalanobis distance is worse than general euclidean, but certainly it fits worse for our sample data. A more fitted distance can save a a lot of algorithm time and gives better results.

Bibliography

Henry E. Brady, David Collier, and Jasjeet S. Sekhon. The neyman-rubin model of causal inference and estimation via matching methods. *The Oxford Handbook of Political Methodology*, 01 2008. doi: 10.1093/oxfordhb/9780199286546.003.0011.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3 edition, 2004.

Richard K. Crump, V. Joseph Hotz, Guido W. Imbens, and Oscar A. Mitnik. Dealing with limited overlap in estimation of average treatment effects. *Biometrika*, 96(1):187–199, 01 2009. ISSN 0006-3444. doi: 10.1093/biomet/asn055. URL <https://doi.org/10.1093/biomet/asn055>.

Colin B. Fogarty, Mark E. Mikkelsen, David F. Gaieski, and Dylan S. Small. Discrete optimization for interpretable study populations and randomization inference in an observational study of severe sepsis mortality. *Journal of the American Statistical Association*, 111(514):447–458, 2016. doi: 10.1080/01621459.2015.1112802. URL <https://doi.org/10.1080/01621459.2015.1112802>.

Jennifer Hill and Yu-Sung Su. Assessing lack of common support in causal inference using bayesian nonparametrics: Implications for evaluating the effect of breastfeeding on childrens cognitive outcomes. *Ann.Appl.Stat.*, 7:1386–1420, 2013. URL <https://projecteuclid.org/euclid.aos/1380804800>.

Guido W. Imbens and Donald B. Rubin. Bayesian inference for causal effects in randomized experiments with noncompliance. *Ann. Statist.*, 25(1):305–327, 02 1997. doi: 10.1214/aos/1034276631. URL <https://doi.org/10.1214/aos/1034276631>.

Gary King and Langche Zeng. The dangers of extreme counterfactuals. *Political Analysis*, 14:131–159, 2006. URL <https://gking.harvard.edu/publications/whatif-software-evaluating-counterfactuals>.

Sharif Mahmood, Hongyuan Lu, and Michael Higgins. Finding common support through largest connected components.

Paul R. Rosenbaum. Optimal matching of an optimally chosen subset in observational studies. *Journal of Computational and Graphical Statistics*, 21:51–71, 2012. URL https://www.jstor.org/stable/23248823?seq=1/subjects#page_scan_tab_contents.

Paul R. Rosenbaum and Donald B. Rubin. The central role of the propensity score in observational studies for causal effects. *Biometrika*, 70(1):41–55, 04 1983. ISSN 0006-3444. doi: 10.1093/biomet/70.1.41. URL <https://doi.org/10.1093/biomet/70.1.41>.

Appendix A

Title for This Appendix

Enter the content for Appendix A in the appendixA.tex file. If you do not have an Appendix A, see comments in the etdrtemplate.tex file for instructions on how to remove this page.

Appendix B

Title for This Appendix

Enter the content for Appendix B in the appendixB.tex file. If you do not have an Appendix B, see comments in the etdrtemplate.tex file for instructions on how to remove this page.