# Proximal Causal Inference with Kernel Methods

**Yibin Xiong** *
University of Southern California
yibinxio@usc.edu

## Abstract

Proximal casual learning is a popular framework to handle unobserved confounding bias. In this project, we focus on using the kernel method under this framework to estimate the average treatment effect non-parametrically, which is probably first proposed by Singh [2020]. I and Kate have digested most parts of the mathematical derivations in [Singh, 2020] and I re-implemented his method in Python with Jax. I applied this method on simulated data and evaluated some metrics on the bias and variance. Challenges and ambiguities in implementation are discussed.

## 1 Introduction

When estimating the average treatment effect (ATE), we need to adjust for the confounders to avoid or relieve confounding bias. However, it is often impossible to identify or collect data on all confounders. Tchetgen et al. [2020] has proposed a general framework called *proximal causal learning*, where proxy variables (also known as negative control [Miao et al., 2018]) are introduced to approximate the effects of unobserved confounders. With some assumptions on the proxy variables, we can screen out the effects of the confounders on the outcome and only keep the causal effects in regression coefficients.

### 1.1 Related Work

Tchetgen et al. [2020] formally introduces the proximal causal learning model. This involves solving an integral equation for the *confounding bridge* function. In the simplest case, the confounding bridge is assumed to be linear and identification of ATE is demonstrated in [Tchetgen et al., 2020]. Nonetheless, this framework is flexible and can be used with non-parametric estimation. Singh [2020] assumes that all functionals are in reproducing kernel Hilbert space (RKHS) and uses the kernel method to identify ATE non-parametrically. He also develop the method in a general way to identify other treatment effects, such as ATE under distribution shift, average treatment effects on treated (ATT), and conditional ATE given that they are all reweighting of the confounding bridge. Although the RKHS theory is a little involved, kernel method provides us not only consistency guarantees but also a neat close-form formulation of how to identify ATE simply through algebraic operations of kernel matrices. The algorithm is essentially a two-stage kernel ridge regression.

Mastouri et al. [2021] also leverages kernel method under proximal causal learning framework. They proposed kernelized proximal variables (KPV) method and proxy maximum moment restriction (PMMR). The former, as far as I am concerned, is the same as Singh's method except how to tune the hyperparameters for regularization in kernel ridge regression. Both of them employ leave-one-out cross validation (LOOCV) with grid search, but they have different formulas. The author of [Singh, 2020] only derives the LOOCV error for the simplified case of regressing one variable to another one, but in the context of proximal causal inference with kernel, we need to regress one variable to *multiple* ones. Thus, it is a little unclear how Singh [2020] tunes the hyperparameter in the second

*I was mentored by Dr. Jie Hu (Kate) on this project from Jun. to Sep. 2022. Kate is a postdoc at Harvard T.H. Chan School of Public Health, Department of Biostatistics.

Preprint. Under review.

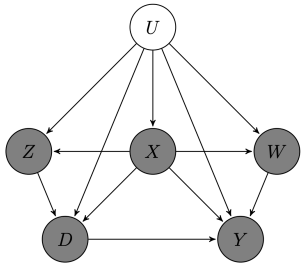


Figure 1: Negative control DAG [Singh, 2020]

stage. In contrast, Mastouri et al. [2021] derives the LOOCV error specifically for the context of KPV.

## 2 Method

In this section, I will review the key assumptions and results in [Singh, 2020]. I will use the following notations. Let $D$ denote the treatment, $X$ be the covariates (usually observed confounders we will adjust), $Z, W$ be the negative control treatment and negative control outcome[2] respectively, $Y$ be the outcome, and $U$ be the unobserved confounders. We want to estimate the ATE $\mathbb{E}\left[Y^{(d)}\right]$ for continuous treatment at value $d$. This is also referred to as dose-response curve in some contexts.

**Assumption 1** (Negative controls)

1. No interference/Consistency: if $D = d$ and $Z = z$, then $Y = Y^{(d,z)}$ and $W = W^{(d,z)}$
2. Latent exchangeability: $\{Y^{(d,z)}\}, \{Z^{(d,z)}\} \perp\!\!\!\perp D, Z \mid U, X$
3. Overlap: If $f(u, x) > 0$, then $f(d, z \mid u, x) > 0$, where $f(u, x)$ and $f(d, z \mid u, x)$ are densities
4. Negative control treatment and outcome: $Y^{(d,z)} = Y^{(d)}$ and $W^{(d,z)} = W$

The first 3 are often assumed for causal inference problems. The last proposition is sometimes written as $Y \perp\!\!\!\perp Z \mid U, X, D$ and $W \perp\!\!\!\perp D, Z \mid U, X$, which means that the negative control treatment $Z$ cannot have causal effects on the outcome $Y$ and the negative control outcome $W$ cannot be causally related to any treatment $D, Z$. Figure 1 shows a DAG of causal relations with negative control variables. This assumption characterizes why $Z$ and $W$ are called negative control variables and are useful for identifying the causal effects in the presence of unobserved confounder $U$.

**Assumption 2** (Confounding bridge)

1. Existence: Let $\gamma_0(d, x, z) := \mathbb{E}[Y \mid D = d, X = x, Z = z]$. There exists a solution $h_0$ to the operator equation

$$\gamma_0(d, x, z) = \mathbb{E}[h(D, X, W) \mid D = d, X = x, Z = z]$$

2. Completeness: For any function $f$,

$$\mathbb{E}[f(U) \mid D = d, X = x, Z = z] = 0 \; \forall \, d, x, z \iff f(U) = 0$$

Existence is the key assumption here while completeness is a technical condition. The purpose of $Z$ is mainly to identify the confounding bridge function while the purpose of $W$ is to surrogate the unobserved confounders when identifying ATE with the confounding bridge.

**Theorem 1** (Identification of ATE)

If Assumptions 1 and 2 hold, then

$$\theta_0^{ATE} = \mathbb{E}_{X,W}[h_0(d, x, w)] = \int h_0(d, x, w) \, d\mathbb{P}(x, w)$$

[2] they are also called treatment-inducing and outcome-inducing proxies

I refer the readers to this note for the proof of this theorem: `http://faculty.washington.edu/yenchic/short_note/note_Proximal.pdf`.

This establishes the general framework of proximal causal learning to identify ATE. Now we can consider the specific case when we use kernel method.

**Conditional mean embeddings** One of the motivations to use kernel is that we can express expectations in the form of inner products of two functions in reproducing kernel Hilbert space (RKHS). For any variable, we can define an RKHS $\mathcal{H}$ with kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that is symmetric positive semi-definite and feature map $\phi = k(x, \cdot)$. The kernel mean embedding for a probability distribution $P$ is defined as $\mu(P) := \int k(x, \cdot)\, dP(x)$, which is a weighted average of the feature map and thus a function in RKHS. By the *reproducing property* of kernel, for any $f \in \mathcal{H} : \langle \mu(P), f \rangle = \mathbb{E}_{X \sim P}[f(X)]$. If $P$ is a conditional distribution, then $\mu(P)$ is called conditional mean embedding.

In proximal causal inference setting, we assume that every variable $D, X, Z, W, Y$ induces an RKHS, whose kernel function is continuous, bounded, and *characteristic* [3]. Then we can express the expectations of our interest into inner products.

**Theorem 2** (identification of ATE with kernel)

$$\gamma_0(d, x, z) = \langle h_0, \phi(d) \otimes \phi(x) \otimes \mu_w(d, x, z) \rangle_{\mathcal{H}}\,, \text{ where } \mu_w(d, x, z) := \int \phi(w)\, d\mathbb{P}(d, x, z) \quad (1)$$

Moreover, given $h_0$,

$$\theta_0^{ATE} = \langle h_0, \phi(d) \otimes \mu \rangle_{\mathcal{H}}\,, \text{ where } \mu := \int [\phi(x) \otimes \phi(w)]\, d\mathbb{P}(x, w) \quad (2)$$

This theorem gives us the identification results when we use kernels. Since we have multiple variables, we work in the tensor product of RKHS's and its feature map is also the tensor product of the feature maps of the single-variable RKHS's. This result outlines the two stages of regression to estimate ATE. In the first stage, we estimate the conditional mean embedding $\mu_w(d, x, z)$ via regression [Singh et al., 2019],[Grünewälder et al., 2012]. In the second stage, we estimate $h_0$ by regressing[4] $\gamma_0(d, x, z)$ to $\phi(d) \otimes \phi(x) \otimes \mu_w(d, x, z)$ according to (1). Finally, we estimate ATE by calculating an inner product of $h_0$ and some other known element. In practice, we work with the kernel matrices, which contains inner products of the feature maps, rather than these functional forms of feature maps.

**Algorithm** Let $\odot$ denotes element-wise product. For stage 1, we use the sample $\{w_i, d_i, x_i, z_i\}_{i=1}^n$. For stage 2, we use a second sample $\{\dot{y}_i, \dot{d}_i, \dot{x}_i, \dot{z}_i\}_{i=1}^m$.

$$\begin{aligned}
&A = K_{DD} \odot K_{XX} \odot K_{ZZ}, \quad && \dot{A} = K_{D\dot{D}} \odot K_{X\dot{X}} \odot K_{Z\dot{Z}} \\
&B = (A + n\lambda I)^{-1}\dot{A}, && M = K_{\dot{D}\dot{D}} \odot K_{\dot{X}\dot{X}} \odot (B^T K_{WW} B) \\
&\hat{\alpha} = (MM^T + m\xi M)^{-1} M\dot{Y}, && \hat{h}(d, x, w) = \hat{\alpha}^T [K_{\dot{D}d} \odot K_{\dot{X}x} \odot (B^T K_{Ww})] \\
&\hat{\theta}^{ATE}(d) = \frac{1}{n}\sum_{i=1}^n \hat{h}(d, x_i, w_i) = \frac{1}{n}\sum_{i=1}^n \hat{\alpha}^T [K_{\dot{D}d} \odot K_{\dot{X}x} \odot (B^T K_{Ww})]
\end{aligned}$$

where $(K_{DD})_{i,j} = k(d_i, d_j)$, $(K_{D\dot{D}})_{i,j} = k(d_i, \dot{d}_j)$ are kernel matrices and $(K_{\dot{D}d})_i = k(\dot{d}_i, d)$ is the kernel evaluation vector. The hyperparameters $\lambda, \xi$ are regularization strength in stage-1 and stage-2 regressions.

Using different samples for the two stages of regression is called *sample splitting*, which helps relieve the finite sample bias. If we want a more efficient use of data, we can use the entire dataset for both stage-1 and stage-2 regression.

[3] the mapping $\mathbb{P} \mapsto \int \phi(x)\, d\mathbb{P}$ is injective

[4] Here involves an analogy to the normal case when the model is $y = w^T x = \langle w, x \rangle$

## 3 Implementation

**Open-source implementation** Unfortunately, there is no public code for [Singh, 2020]. I requested the author for code and he has a simple R code that is not scalable to large data. The authors of [Mastouri et al., 2021] have released their code[5], but the code is not well-organized and not very interpretable for reproducing the results with data from a different simulation model. Recently Kate has found another paper [Kompa et al., 2022] that builds on the idea of maximum moment restriction, which is another track of proximal learning that we have not fully explored in this project, and has a better organized code[6] of KPV method in [Mastouri et al., 2021] because they use it as a baseline. The contribution of this paper to our project is their implementation of KPV. However, I failed to run their code on our simulated data because the main purpose of their repository is to replicate numerical experiments in their paper [Kompa et al., 2022]. It might be usable but requires us to take a more detailed inspection.

**JAX** I implemented the algorithm in Python. To utilize GPU for faster computation, I adapted my original code in NumPy to the JAX package, which integrates NumPy functions with GPU. This boosts the runtime from 62s to 6s when there is no sample splitting and the sample size is 1000. All experiments are run on Google Colab with Tesla T4 GPU.

**My trick to boost runtime** When we do grid search to tune the hyperparameters, we plug in different values of $\lambda$ and $\xi$ to the formula of LOOCV error keeping other things the same. Thus, in the tuning of $\lambda$, it is wasteful to do matrix inversion in $(A + n\lambda I)^{-1}A$ for multiple times. Instead, I computed the eigen-decomposition of $A := PDP^T$ at the beginning and each time we just calculate

$$(A + n\lambda I)^{-1}A = P(D + n\lambda I)^{-1}DP^T = P(D/(D+n\lambda))P^T$$

where the addition and division (in red) are performed element-wise only on the diagonal elements. In this way we uses matrix multiplications to substitute costly matrix inversions. Refraining from inverting large matrices also sidesteps some numerical problems (see section 5). In addition, in the tuning algorithm, some matrices we calculated will be reused later in the main algorithm. I stored these matrices to avoid repeated work. For the code to calculate the kernel matrices and kernel evaluation vectors/matrices, I tried to vectorize them as much as possible using NumPy's broadcasting mechanism.

## 4 Simulation Study

**Model** Kate suggested a slightly modified version of the simulation model in [Singh, 2020] to produce synthetic data.

1. Draw unobserved noise $\{\epsilon_i\}_{i\in[2]} \overset{i.i.d.}{\sim} \mathcal{N}(0.5, 1)$, $\{\nu_i\}_{i\in[2]} \overset{i.i.d.}{\sim} U(-1, 1)$
2. Unobserved confounders $u_1 = \epsilon_1$ (alteratively, $u_1 = \epsilon_1 + \epsilon_2$)
3. Negative controls $Z = \nu_1 + 0.25 \cdot u_1, W = \nu_2 + 0.25 \cdot u_1 + 0.45 \cdot u_1^2$
4. Covariates $X \sim \mathcal{N}(0, \Sigma)$, where $\Sigma \in \mathbb{R}^{p\times p}$ s.t. $\Sigma_{ii} = 1$ and $\Sigma_{ij} = \frac{1}{2} \cdot 1\{|i - j| = 1\}$ for $i \neq j$
5. Treatment $D = \Phi(3X^T\beta) + 0.5 \cdot u_1$, where $\beta \in \mathbb{R}^p$ s.t. $\beta_j = j^{-2}$
6. Outcome $Y = 1.2(D + X^T\beta) + D^2 + DX_1 + 0.5 \cdot u_1$

The difference is that we made $Z$ and $W$ not causally related to $D$ and $Y$, respectively. This means in the formulas for $D$ and $Y$, there should be no $Z$ and $W$. They can still be correlated with $D$ and $Y$ through the confounders and their sole effect is to approximate the unobserved confounders.

The number of covariates, i.e. $p$ is set to 5. To evaluate the variance of our estimation, we need to have multiple experiments. Thus, I conducted 20 trials as Singh [2020] did. In each trial, if I use sample splitting, the data are split into two halves (i.e. `split_ratio = 0.5`) and each of them is used for a stage of regression. If there is no sample splitting, the entire dataset is used repeatedly

[5] `https://github.com/yuchen-zhu/kernel_proxies`

[6] `https://github.com/beamlab-hsph/Neural-Moment-Matching-Regression`

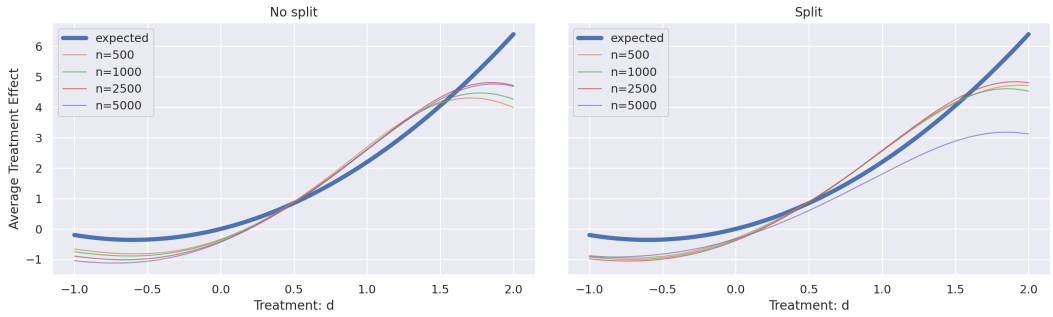


Figure 2: Fitted dose-response curve

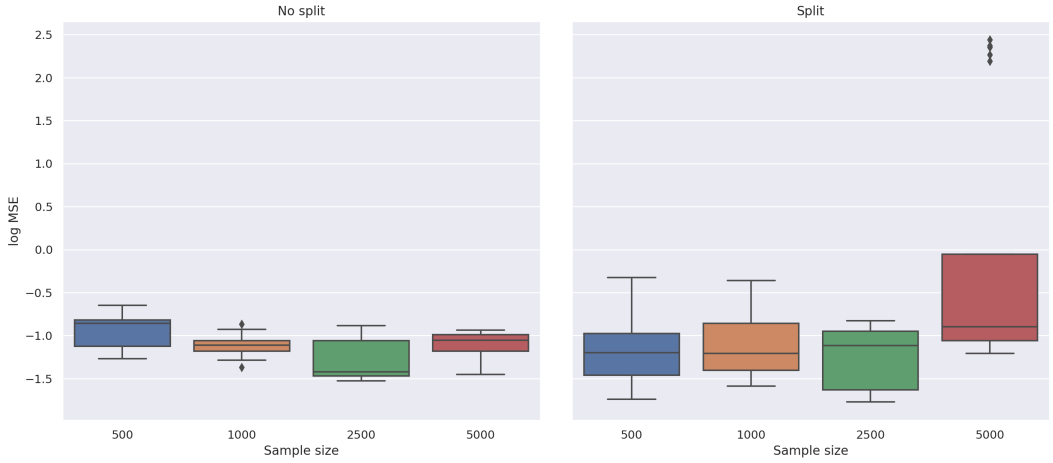


Figure 3: log MSE

in both stages. To explore the convergence rate of the finite sample bias, I gradually increase the sample sizes $n, m$ $(n = m)$ from 500 to 10000. I generated the full dataset of 10000 examples first. For non-splitting experiments, I take the first $n$ out of 10000 examples as the dataset. For splitting experiments, I take the first $n + m$ out of 10000 examples and split the dataset into two.

The treatment $D$ in this model is approximately supported on $[-1, 2]$. Therefore, I estimated the ATE at 301 evenly spaced points between -1 and 2. For our model, it is easy to figure out that the ATE $\mathbb{E}[Y^{(d)}] = 1.2d + d^2$.

**Results** Figure 2 shows the fitted curves of ATE averaged over 20 trials. We don't see sample splitting has a distinct margin in the performance over non-splitting setting. Note that for sample splitting, the mean estimate with sample size $n = m = 5000$ has a worse performance than using smaller sample sizes. This might be related to numerical problems when dealing with large matrices. Section 5 discusses this issue in detail.

To determine how robust our algorithm is, I evaluated the log mean squared error (MSE) for each trial (i.e. mean is taken over the 301 treatment values at which we I evaluate ATE) and plot the distributions of the 20 log MSE's in a box-plot (Figure 3). MSE contains both bias and variance of the estimate. The experiments without sample splitting generally has smaller log MSE than their counterparts at every sample size. Again, there are abnormal estimation results when $n = 5000$, which produces an error much larger than fitting a smaller dataset does. Empirically, I have not found any advantage of sample splitting.

In terms of the runtime, since there are matrix multiplications and matrix inversions in tuning the hyperparameter $\xi$, it is expected to take $\mathcal{O}(n^3)$ time. Figure 4 shows the empirical running time and the log curve on the right confirms a polynomial growth. For `num_trials = 20`, `n = 5000`, our algorithm only ran for 143s in sample splitting setting, which is promising for scaling up to larger datasets. However, when I increased sample size to $n = 10000$, there is out-of-memory error on Google Colab.

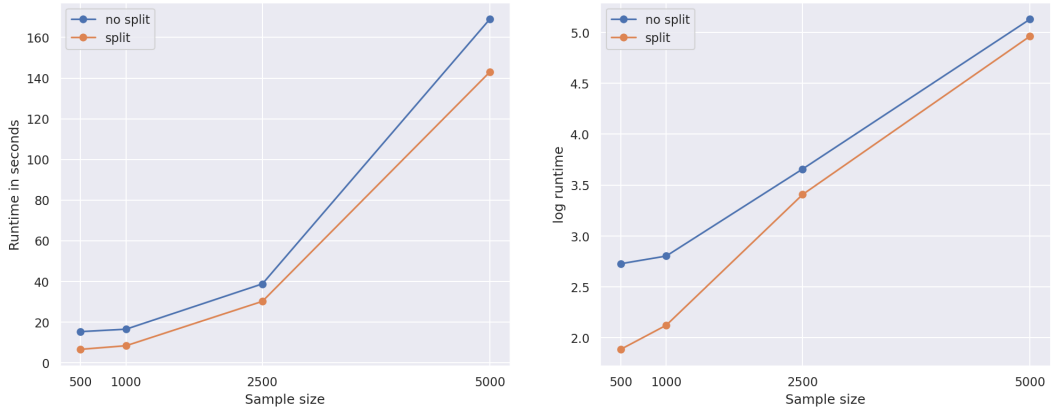


Figure 4: Runtime

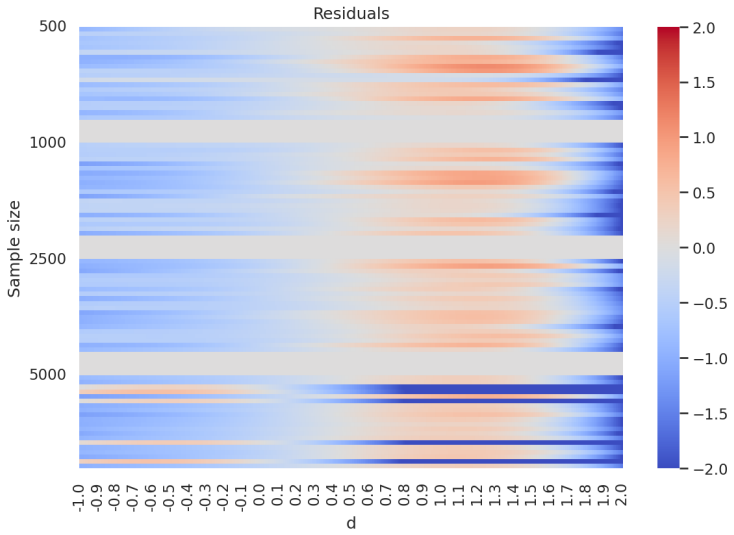


Figure 5: Residuals of Different Trials (with splitting)

## 5 Discussions

**Numerical problems** When working with large matrices, it is common to encounter some matrices that are very close to singular that we need to invert, which severely affects the results. In our case, we need to invert $M + n\xi I$ when doing grid search for $\xi$. As sample size increases, $M$ is numerically much closer to singular and adding a small regularization will not relieve the ill condition. To verify this, I printed the smallest eigenvalue of $M$ and it is `2.4750e-06` when $n = 5000$ and `6.0248e-04` when $n = 2500$. Inverting ill-conditioned matrices will result in insanely large values, severely harm the final estimates, and mislead the algorithm to choose a bad hyperparameter. Choosing large enough regularization strength (i.e. the range for grid search) is a straightforward solution, but in practice it does not help much and will increase finite-sample bias of the estimates. If this range is not well chosen, there will be `nan` for the LOOCV error that we want to minimize.

Figure 5 is a diagnostic plot that enables us to see which trial among the 20 has large residuals from the ground-truth at which treatment value. Each strip corresponds to a trial and trials with different sample sizes are separated. We can see that for $n = m = 5000$, there are several trials that produces completely wrong estimates in the range of $[0.6, 2]$. I am not completely sure that inverting ill-conditioned matrices is the culprit of bad performance because a significant proportion of the trials have normal performance.

**Future Work** The numerical problem should be clearly diagnosed and solved. Otherwise it will be more obvious with larger input data. If the algorithm works well on simple, simulated models, I and Kate planned to apply it to simulated time series data and real-world longitudinal data to investigate the causal effects of

temperature on hospitalization. Time naturally induces some negative control variables. For example, the $SO_2$ level in the next day cannot be causally related to the hospitalization on the current day, so it can be used as a negative control treatment. See common examples of how to develop negative control variables in [Shi et al., 2020]. If KPV method does not work well, we can explore the family of maximum moment restriction (MMR) methods and leverage neural networks [Kompa et al., 2022], which has better expressiveness than kernels and can fit any function better. In terms of scalability, my implementation suffers from the constraint of memory. We might need to figure out how to run the public repositories on our data or use better devices than Google Colab free GPU.

# References


Rahul Singh. Kernel methods for unobserved confounding: Negative controls, proxies, and instruments. *arXiv preprint arXiv:2012.10315*, 2020.

Eric J Tchetgen Tchetgen, Andrew Ying, Yifan Cui, Xu Shi, and Wang Miao. An introduction to proximal causal learning. *arXiv preprint arXiv:2009.10982*, 2020.

Wang Miao, Xu Shi, and Eric Tchetgen Tchetgen. A confounding bridge approach for double negative control inference on causal effects. *arXiv preprint arXiv:1808.04945*, 2018.

Afsaneh Mastouri, Yuchen Zhu, Limor Gultchin, Anna Korba, Ricardo Silva, Matt Kusner, Arthur Gretton, and Krikamol Muandet. Proximal causal learning with kernels: Two-stage estimation and moment restriction. In *International Conference on Machine Learning*, pages 7512–7523. PMLR, 2021.

Rahul Singh, Maneesh Sahani, and Arthur Gretton. Kernel instrumental variable regression. *Advances in Neural Information Processing Systems*, 32, 2019.

Steffen Grünewälder, Guy Lever, Luca Baldassarre, Sam Patterson, Arthur Gretton, and Massimilano Pontil. Conditional mean embeddings as regressors-supplementary. *arXiv preprint arXiv:1205.4656*, 2012.

Benjamin Kompa, David R Bellamy, Thomas Kolokotrones, James M Robins, and Andrew L Beam. Deep learning methods for proximal inference via maximum moment restriction. *arXiv preprint arXiv:2205.09824*, 2022.

Xu Shi, Wang Miao, and Eric Tchetgen Tchetgen. A selective review of negative control methods in epidemiology. *Current epidemiology reports*, 7(4):190–202, 2020.