Lab 6: Bayesian Estimation of Causal Effects with Bayesian Additive Regression Trees

Alex Ziyu Jiang

```r
# install and load the packages
#install.packages("BayesTree")
#install.packages("LaplacesDemon")
#install.packages("latex2exp")

library(BayesTree)
library(rstan)
```

```
## Loading required package: StanHeaders

## Loading required package: ggplot2

## rstan (Version 2.21.3, GitRev: 2e1f913d3ca3)

## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
```

```r
library(LaplacesDemon)
library(latex2exp)
```

```
## Warning: package 'latex2exp' was built under R version 4.1.2
```

(Code example in this lab is adapted from https://github.com/stablemarkets/intro_bayesian_causal)

A quick review

Before we dive into the meat of the lecture today let's have a quick warmup on Bayesian causal effect estimation. To answer a causal question it will be helpful to consider the three following question first:

1. **What is the data? Is it from an experimental design, or observational studies?** Recall the following notations we use in class:

- $Y_{1:n}$: the **observed** outcome variable
- $X_{1:n}$: the treatment variable (for simplicity, X is binary for the model we consider.)
- $\mathbf{Z}_{1:n}$: confounder/convariates

2. **What is the causal quantity we are interested in?** We would want to conduct estimation on the average treatment effect(ATE):

$$\text{ATE} = E\left[Y_i(1)\right] - E\left[Y_i(0)\right]$$

We know from the lecture that without further assumptions this is not identifiable and estimable due to the fact that we do not actually observe the potential outcomes.

3. **What are the causal assumptions that fill the gap between the data we have and the quantity we would like to estimate?**

Two important assumptions:

- Conditional ignorability: $Y_i(x) \perp X_i \mid Z_i$
- Consistency: $Y_i = X_i Y_i(1) + (1 - X_i)\, Y_i(0)$ (for binary treatments)

Based on the two assumptions above we can estimate the ATE using our data:

$$\begin{aligned}
E\left[Y_i(x)\right] &= \sum_z E\left[Y_i(x) \mid Z_i = z\right] P\left(Z_i = z\right) \\
&= \sum_z E\left[Y_i(x) \mid X_i = x, Z_i = z\right] P\left(Z_i = z\right) \\
&= \sum_z E\left[Y_i \mid X_i = x, Z_i = z\right] P\left(Z_i = z\right) \\
&= E\left[E\left[Y_i \mid X_i = x, Z_i\right]\right]
\end{aligned}$$

thus

$$ATE = E_Z\left[E_{Y|Z,X}\left[Y_i \mid X_i = 1, Z_i\right] - E_{Y|Z,X}\left[Y_i \mid X_i = 0, Z_i\right]\right]$$

From the formula above, we see that to estimate ATE we have two tasks:

1. Estimate the outer expectation with respect to Z: this can be done by Bayesian bootstrapping(BB). We learned how to do this in class.
2. Estimate the inner expectation with respect to Y given X, Z: this requires us to define a model of $Y \mid X, Z$. We learned to frame $Y \mid X, Z$ as a Bayesian linear model in class.

BART (Bayesian Additive Regression Trees)

Previously we learned to model $E[Y \mid X, Z]$ as a parametric models, these models are easy to implement and readily interpretable, but they also place very stringent restrictions on the form of the regression functions. To model the usually nonlinear treatment and covariate effects and to characterize the interactions between these two variables, we need to consider other flexible modeling techniques.

- One choice is to use spline models. This way we stay parametric but we will be able to fit models that have more curvature and may perform better on wiggly data. The framework is very similar to linear models.
- Otherwise, we can consider Bayesian nonparametric models(BNP). In simple words, the parameter space for BNP models have inifinite dimensions and it usually allows for great flexibility in modeling.
- Today we will introduce Bayesian Addition Regression Trees as a common BNP method to model $E[Y \mid X, Z]$.

Model description

Essentially, BART defines a step function for the expected outcome $E[Y \mid X, Z]$, defined on the covariate space spanned by X, Z. We first consider this single tree model (for simplicity, I will use X to represent the 'covariates' X, Z in this section):

$$\mu(X) = g\left(X; T, M\right)$$

You can think of T as a series of 'splitting rules' that separates the covariate space of X into a couple of disjointed areas, in a way that for each possible set of covariate $X = x$, it belongs to one (and only one) of these areas. Finally, each area is assigned a 'node value' (controlled by M) as the function outcome for all X that falls into that particular area.

In practice, we usually model the expected outcome as a sum of different 'trees':

$$\mu(X) = \sum_{j=1}^{J} g\left(X; T_j, M_j\right)$$

Simultaneously fitting multiple trees tend to give results that are more flexible, and each tree tend to be 'shallower', and thus helping prevent model fitting.

Tree Prior

As I mentioned before, essentially we can think of T as a series of splitting rules – there are several aspects to think about. To define prior for T, what we need to do is to assign these 'splitting rules' prior probabilities, as a way of saying 'what type of splitting rule is more likely' based on our belief. To give a more comprehensive description of 'splitting rule', we need to consider the following aspects:

- 'Whether to split' – this correspond to the 'tree structure' of T, i.e. whether will a node be split into children at a deeper level? This is controlled by two hyperparameter, in that a node at depth $d = 0, 1, 2, ...$ will be split is $\frac{\alpha}{(1+d)^{\beta}}$. We can see that larger α and smaller β will lead to bigger and deeper trees (and vice versa).
- After deciding to split a node, we need to decide which covariate in X we should split on. Usually we just fit a uniform prior over all possible covariates.
- After deciding on splitting and the covariate to be splitted, we then determine the threshold value – usually this is determined by a uniform prior over all the data in that covariate variable we chose.

Other Details on BART

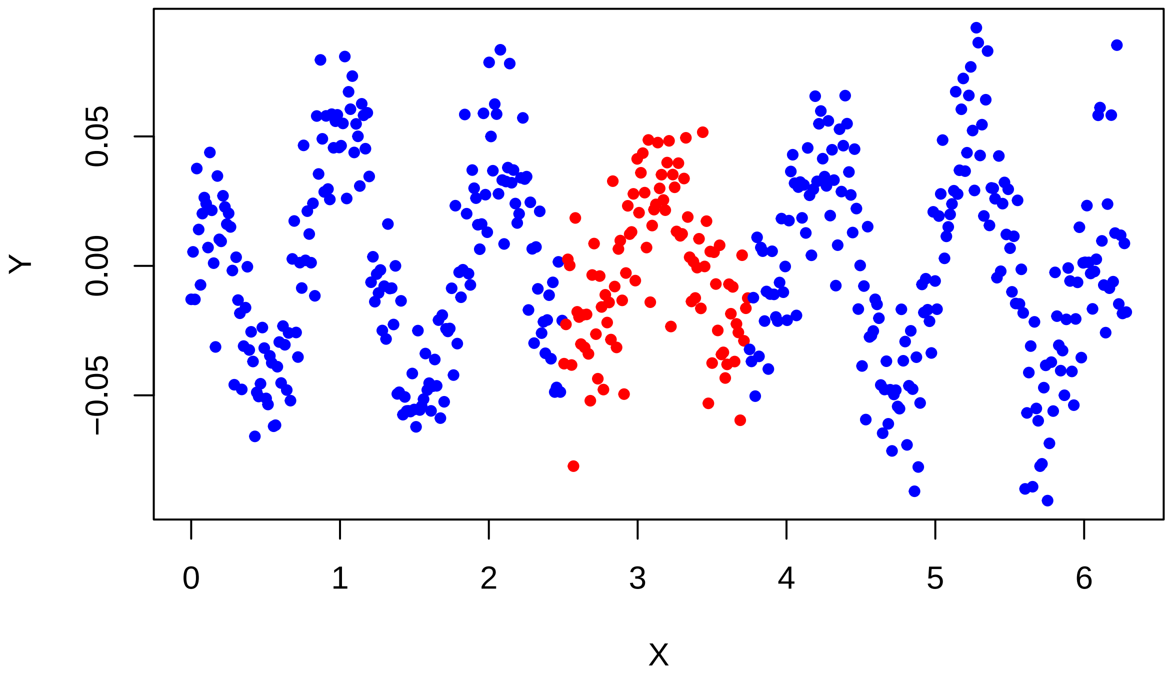
- Much like the regression coefficients in linear models, we usually place normal priors on the node parameters M for conjugacy. Also for the residual variance parameter σ, we place an inverse Gamma prior.
- We will not get into the details about how to implement BART in this course, but to give you a flavor of it, we essentially create a Markov Chain of 'Trees', and for each update, we do some 'pruning' to the tree – we either grow a node, kill a pair of leaf nodes, or change some splitting rules within each node. This part is done by the Metropolis-Hastings algorithm, which you might have learned in class.
- The other parameters can be updated through a relatively straightforward posterior Gibbs sampling procedure.
- A trick called **Backfitting method** is used for updates with multiple trees – essentially it allows you to update one tree at a time and makes the tree update simpler

A simulation example

Here we look at how BART performs on actual data. Here the systematic relationship between Y and X is a highly nonlinear function composed of exponential functions and cosine functions. A residual standard deviation of 0.02 is added to the data. Here we choose the data with X between 2.5 and 3.75 as the test set. We first visualize the data:

```
set.seed(1)
N = 500
X = seq(0, 2*pi, length.out = N)
X_01 = ((X - min(X)) / ( max(X) - min(X) ))
```

```
oscil1 = exp(-1*X)*cos(2*pi*X)
oscil2 = rev((exp(-1*X)*cos(2*pi*X)))
Y = (X_01)*oscil1 + (rev(X_01))*oscil2  + rnorm(N,0, .02)
test = ifelse(X>2.5 & X<3.75, 1, 0)
plot(X, Y, col=ifelse(test==1, 'red', 'blue'), pch=20)
```



```
d = data.frame(X=scale(X), Y=scale(Y))
d_train = d[test==0, ]
d_test = d[test==1, ]
```

We then fit the data to a BART model and inspect the results. Implementation of BART is complicated but luckily we can use `BayesTree` as a built-in package to aid our computation. We can tune the number of trees, the tree hyperparameters to see how it affect the results. Notice that BART is actually not very good at 'pick up' the non-linear trend in the dataset in the middle, as the tree structure of the model produces a more rigid result – we will learn models that are more flexible next week.

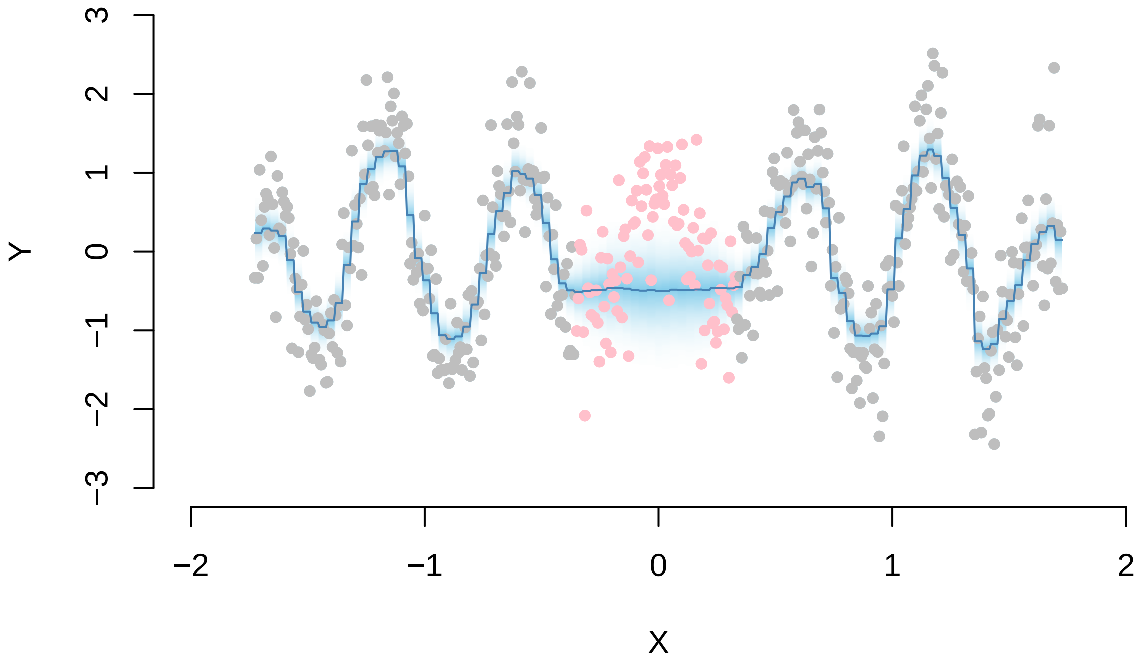
The reason for this, is because the nonparametric methods do not guarantee to recover the true function, the are good at local approximations based on the data near by, but when we move outside from the support of our training data (i.e. doing **extrapolation**), we do not have any guarantee for our prediction accuracy without further assumptions. In the figure above, the red data in the middle could be anywhere, but our models wouldn't be able to pick up the trends.

```
set.seed(3)
bart_res = bart(x.train = d_train$X, y.train = d_train$Y, x.test = d_test$X,
                ntree=200, # number of trees
                power = 3, # tree prior, smaller power -> more nodes
                base = 0.95 # between 0 and 1, larger base -> more nodes
                )
```

```
##
##
## Running BART with numeric y
##
## number of trees: 200
## Prior:
##  k: 2.000000
##  degrees of freedom in sigma prior: 3
##  quantile in sigma prior: 0.900000
##  power and base for tree prior: 3.000000 0.950000
##  use quantiles for rule cut points: 0
## data:
##  number of training observations: 401
##  number of test observations: 99
##  number of explanatory variables: 1
##
##
## Cutoff rules c in x<=c vs x>c
## Number of cutoffs: (var: number of possible c):
## (1: 100)
##
##
## Running mcmc loop:
## iteration: 100 (of 1100)
## iteration: 200 (of 1100)
## iteration: 300 (of 1100)
## iteration: 400 (of 1100)
## iteration: 500 (of 1100)
## iteration: 600 (of 1100)
## iteration: 700 (of 1100)
## iteration: 800 (of 1100)
## iteration: 900 (of 1100)
## iteration: 1000 (of 1100)
## iteration: 1100 (of 1100)
## time for loop: 15
##
## Tree sizes, last iteration:
## 3 2 2 2 3 2 3 2 2 1 3 2 2 2 3 2 4 2 2 3
## 2 3 3 2 2 2 2 3 2 2 2 1 3 1 2 2 2 2 2 3
## 2 2 2 2 4 2 3 2 2 2 2 2 2 2 2 2 2 2 3 3
## 4 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 3 2 2
## 2 1 2 3 2 3 2 2 3 1 2 2 2 2 2 2 2 2 2 2
## 2 2 4 2 2 2 2 2 2 2 2 2 3 3 3 2 2 2 3 2
## 2 2 2 2 2 2 3 2 2 3 1 1 2 2 2 2 1 2 2 2
## 3 3 2 2 2 3 2 2 3 3 3 2 3 3 2 2 2 2 2 2
## 2 3 3 2 2 3 3 2 2 2 2 2 3 2 2 2 2 2 3 2
## 2 2 4 4 2 2 2 2 2 2 3 3 2 2 2 3 3 2 2 2
## Variable Usage, last iteration (var:count):
## (1: 247)
## DONE BART 11-2-2014
```

```r
#png('bart_oscil.png')
plot(d$X, d$Y, pch=20,
     xlim=c(-2,2), ylim=c(-3,3),
     col=ifelse(test==1,'pink','lightblue'),
     axes=F, xlab='X', ylab='Y')

### Plot posterior credible Band
colfunc <- colorRampPalette(c("white", "skyblue"))
```

```r
ci_perc = seq(.99,.01,-.01)
colvec = colfunc(length(ci_perc))
names(colvec) = ci_perc

X_draw = c(d_train$X, d_test$X)
ord = order(X_draw)

X_draw = X_draw[ord]
res_stack = rbind(t(bart_res$yhat.train), t(bart_res$yhat.test))
res_stack = res_stack[ord,]

for(i in ci_perc){
  pci = apply(res_stack, 1, quantile, probs=c( (1-i)/2, (1+i)/2  ) )
  polygon(c(X_draw,rev(X_draw)),c(pci[1,],rev(pci[2,])),
          col = colvec[as.character(i)], border = F)
}

axis(1, -2:2, -2:2)
axis(2, -3:3, -3:3)

points(d$X, d$Y, pch=20, col=ifelse(test==1,'pink','gray'))
points(d$X, rowMeans(res_stack), col='steelblue', pch=20, type='l', lwd=1)
```



```r
#dev.off()
```

Estimating ATE - an example

Here we consider a simulation example where we estimate the ATE using the backdoor formula. We consider a case where the model is simulated as follows:

- We have one confounding variable Z. We sample it from a normal distribution $N(0, 1)$

- Given the confounder $Z_i = z_i$, the probability of observation i being given treatment is

$$P(X_i \mid Z_i = z_i) = \frac{1}{1 + \exp(0.5Z_i - 1)}$$

- The response Y_i, given the confounder and treatment variable, follows a normal distribution $N((z_i + 0.5z_i^2)X_i, 0.04)$ (i.e. observations in the control group has zero mean, but the ones in the treatment group has mean following a nonlinear relationship with the confounder).

Here we see that $E[Y \mid X, Z] = X(Z + 0.5Z^2)$ – this is a highly nonlinear relationship with respect to X, Z that is not linearly separable. BART can be used to 'decipher' such nonlinearity.

Recall that under our BART model, we have $E[Y \mid X, Z] = \sum_{j=1}^{J} g\left(X, Z; T_j, M_j\right)$. Note that both X and Z are treated as the 'covariates' in the model here (which makes sense, because we want to discover the nonlinearity between these two).

After running the Markov Chain Monte Carlo algorithm that implements the BART model, what we get is a series of posterior samples of the tree and node parameters: $\{T_j^{(m)}, M_j^{(m)}\}_{m=1}^{M}$, this allows us to compute the M posterior draws of the expected outcome:

$$\mu^{(m)}(X, Z) = \sum_{j=1}^{J} g\left(X, Z; T_j^{(m)}, M_j^{(m)}\right)$$

Conditioning on Z, the posterior draws of conditional ATE can be expressed as

$$\mu^{(m)}(1, Z) - \mu^{(m)}(0, Z)$$

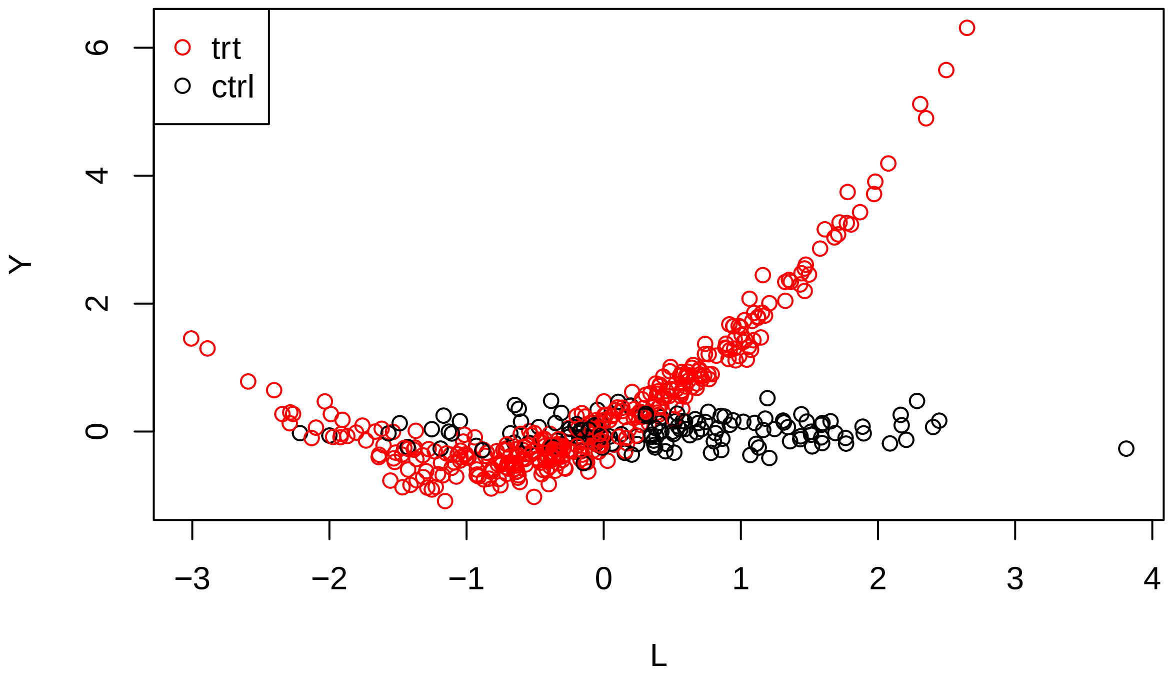
finally we integrate out Z to get posterior draws of ATE – this is done by Bayesian Bootstrapping.

$$\Psi^{(m)} = \sum_{i=1}^{n} p_i^{(m)} \left(\mu^{(m)}(1, Z_i) - \mu^{(m)}(0, Z_i)\right)$$

where $(p_1, ..., p_n)$ is drawn from a Dirichlet distribution of parameters $(1, 1, 1, ..., 1)$. With the posterior draws of the ATE, we can calculate its posterior mean, credible intervals and etc.

```r
bayes_boot = function(mu_a1, mu_a0){
  n = nrow(mu_a1)
  M = ncol(mu_a1)
  psi_post = numeric(M)

  for(m in 1:M){
    bb_weights = rdirichlet( 1, rep(1, n) )
    psi_post[m] = sum(bb_weights*( mu_a1[, m] - mu_a0[ , m] ))
  }

  return(psi_post)
}

#### ------------------------ Simulate Data --------------------------------####
set.seed(1)

N = 500
L = rnorm(N)
A = rbinom(N, 1, invlogit(1 - .5*L ) )
Y = rnorm(N, 0 + (L + .5*L^2 )*A , .2)
```

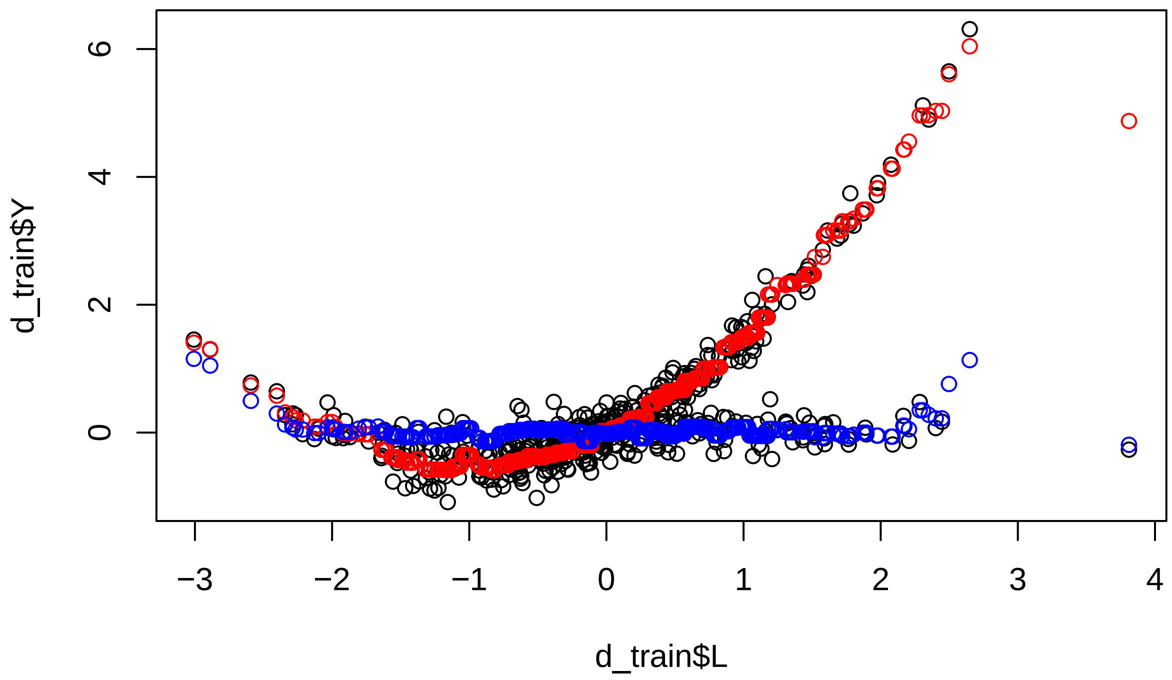
```r
plot(L, Y, col=ifelse(A==1, 'red','black'))

d_train = data.frame(Y=Y, A=A, L=L )

# testing data: for treatment and control, we want predictions
# on every L

d_a1 = data.frame(A=1, L=d_train$L)
d_a0 = data.frame(A=0, L=d_train$L)

d_test = rbind(d_a1, d_a0)
legend("topleft", legend = c("trt","ctrl"), col = c("red", "black"), pch = 1)
```



```r
#### ----------------------- BART Model          ----------------------------####
set.seed(3)
bart_res = bart(x.train = d_train[, c('L','A') ], y.train = d_train$Y,
                x.test = d_test, ndpost = 500, nskip = 500)
```

```
##
##
## Running BART with numeric y
##
## number of trees: 200
## Prior:
##  k: 2.000000
##  degrees of freedom in sigma prior: 3
##  quantile in sigma prior: 0.900000
##  power and base for tree prior: 2.000000 0.950000
##  use quantiles for rule cut points: 0
## data:
##  number of training observations: 500
##  number of test observations: 1000
##  number of explanatory variables: 2
##
##
## Cutoff rules c in x<=c vs x>c
## Number of cutoffs: (var: number of possible c):
## (1: 100) (2: 100)
##
##
## Running mcmc loop:
## iteration: 100 (of 1000)
## iteration: 200 (of 1000)
## iteration: 300 (of 1000)
## iteration: 400 (of 1000)
## iteration: 500 (of 1000)
## iteration: 600 (of 1000)
## iteration: 700 (of 1000)
## iteration: 800 (of 1000)
## iteration: 900 (of 1000)
## iteration: 1000 (of 1000)
## time for loop: 22
##
## Tree sizes, last iteration:
## 2 1 2 2 2 2 2 2 2 3 3 3 3 2 2 2 2 2 3 2
## 2 2 3 2 2 4 2 2 2 2 2 2 2 2 2 2 2 2 2 3
## 2 2 3 3 2 4 3 3 3 2 4 2 2 2 2 2 3 1 2 4
## 2 2 2 1 1 2 2 2 2 2 2 4 2 4 3 3 2 2 2 2
## 4 2 3 2 2 2 3 2 2 3 2 1 4 2 2 2 2 4 2 2
## 4 2 2 2 2 2 2 2 2 2 3 2 2 3 4 2 2 3 3 2
## 3 2 2 3 2 2 2 1 2 2 2 2 1 2 3 1 2 2 2 1
## 2 3 2 3 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 3
## 2 2 2 2 2 2 2 2 3 2 2 3 2 2 2 1 2 3 2 2
## 3 2 2 2 2 4 1 2 2 3 2 2 2 2 2 1 2 2 2 2
## Variable Usage, last iteration (var:count):
## (1: 129) (2: 117)
## DONE BART 11-2-2014
```

```r
bart_res_pred = t(bart_res$yhat.test)

mu_a1 = bart_res_pred[1:N, ]
mu_a0 = bart_res_pred[(N+1):(2*N), ]

psi_bart = bayes_boot( mu_a1 = mu_a1, mu_a0 = mu_a0)

plot(d_train$L, d_train$Y)
points(d_train$L, rowMeans(mu_a1), col='red')
points(d_train$L, rowMeans(mu_a0), col='blue')
```



##

Bayesian Linear Model

As a comparison we can see how a parametric model will fare on this setting. We consider a Bayesian linear regression with very flat priors on the model parameters:

$$
\begin{aligned}
E[Y_i \mid X_i, Z_i] &= \beta_0 + \beta_X X_i + \beta_L Z_i \\
\beta_0, \beta_X, \beta_L &\sim N(0,4) \\
\sigma &\sim Unif(0,50)
\end{aligned}
$$

Under the linear model the conditional ATE is just the regression coefficient that corresponds to X, β_X, we extract the posterior samples from the model and do the Bayesian bootstrapping:

$$\Psi_{LM}^{(m)} = \sum_{i=1}^{n} p_i^{(m)} \beta_X^{(m)}$$

```
#### ----------------------- Linear Regression (Bayesian)   --------------------------####
library(rjags)
```

```
## Loading required package: coda

##
## Attaching package: 'coda'

## The following object is masked from 'package:rstan':
##
##     traceplot

## Linked to JAGS 4.3.0

## Loaded modules: basemod,bugs
```

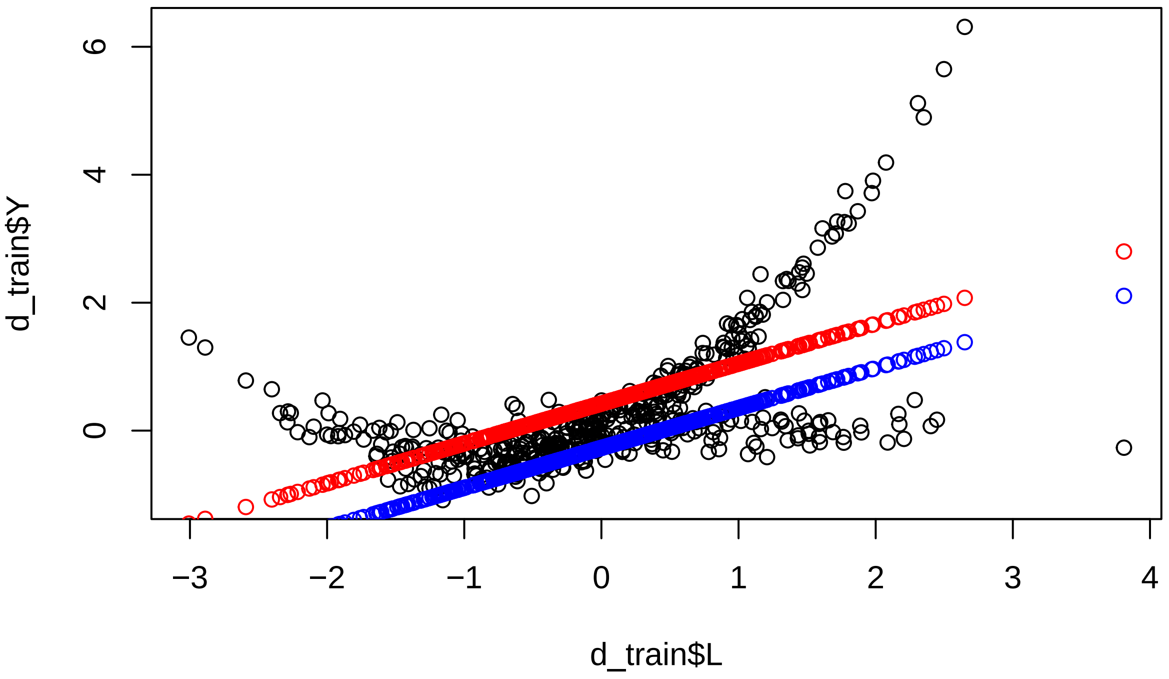
```
## compute bootstrap distribution for psi_lm
set.seed(123)
B=500
```

```r
psi_bayeslm = numeric(B)

linear_model_code <- "
  data{
    D <- dim(y)
    n <- D[1]
  }
  model{

   for(i in 1:n){
      # likelihood
      y[i] ~ dnorm(mu[i], tau)

      # posterior predictive
      ynew[i] ~ dnorm(mu[i], tau)
   }

    # conditional mean using matrix algebra
    mu <- beta_0 + beta_A*A + beta_L*L
    beta_0 ~ dnorm(mb, pow(sb,-2))
    beta_A ~ dnorm(mb, pow(sb,-2))
    beta_L ~ dnorm(mb, pow(sb,-2))
    sigma ~ dunif(0, smax)
    tau <- pow(sigma, -2)
  }
"
m3 <- jags.model(file = textConnection(linear_model_code),
                 data = list(y = d_train$Y,
                             A = d_train$A,
                             L = d_train$L,
                             mb = 0,
                             sb = 2,
                             smax = 50))
```

```
## Compiling data graph
##    Resolving undeclared variables
##    Allocating nodes
##    Initializing
##    Reading data back into data table
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 500
##    Unobserved stochastic nodes: 504
##    Total graph size: 2519
##
## Initializing model
```

```r
M <- 1e3
m3.samples <- jags.samples(m3,
                           variable.names = c("beta_0", "beta_A", "beta_L"),
                           n.iter = M)
beta_A_post <- as.matrix(m3.samples$beta_A)
```

```r
beta_L_post <- as.matrix(m3.samples$beta_L)
beta_0_post <- as.matrix(m3.samples$beta_0)
mu_a1 <- matrix(0, nrow = N, ncol = M)
mu_a0 <- matrix(0, nrow = N, ncol = M)
for (i in 1:N) {
  mu_a1[i,] <- beta_0_post + beta_A_post + beta_L_post*d_train$L[i]
  mu_a0[i,] <- beta_0_post + beta_L_post*d_train$L[i]
}
psi_bayeslm = bayes_boot( mu_a1 = mu_a1, mu_a0 = mu_a0)
plot(d_train$L, d_train$Y)
points(d_train$L, rowMeans(mu_a1), col='red')
points(d_train$L, rowMeans(mu_a0), col='blue')
```



We see that the prediction from the linear model has a large bias here, due to the linearity constraints in the model. ## A Frequentist Approach Used in Paper

The paper also proposed a frequentist approach for fitting the linear model. I believe it make more sense to compare between Bayesian methods, and that the Bayesian linear model fit should be relatively similar to the frequentist result, under a flat prior on the parameters. But here I include the frequentist procedure for completeness:

- We first estimate $\beta_0, \beta_X, \beta_L$ using OLS
- As we are using a frequentist approach here, we respectively adopt the frequentist bootstrap method. This requires us to sample B pseudodatasets from the original dataset with replacement
- For each pseudodataset, we calculate the estimated ATE based on the backdoor estimation formula. This allow us to get B bootstrap samples of ATE.
- This allows us to calculate mean and quantiles for the bootstrap samples, which can be compared to our results from the BART estimate

```r
#### ----------------------- Linear Regression    --------------------------####

## compute bootstrap distribution for psi_lm
set.seed(123)
B=500
```

```r
psi_lm_boot = numeric(B)
for(b in 1:B){
  id = sample(1:N, size = N, replace = T)
  d_train_b = d_train[id,]
  d_a1_b = data.frame(A=1, L=d_train_b$L)
  d_a0_b = data.frame(A=0, L=d_train_b$L)
  d_test_b = rbind(d_a1_b, d_a0_b)
  lm_b = lm(data = d_train_b, Y ~ A + L)
  pred_lm_b = predict(lm_b, d_test_b)
  psi_lm_boot[b] = mean(pred_lm_b[1:N] - pred_lm_b[(N+1):(2*N)])
}
```

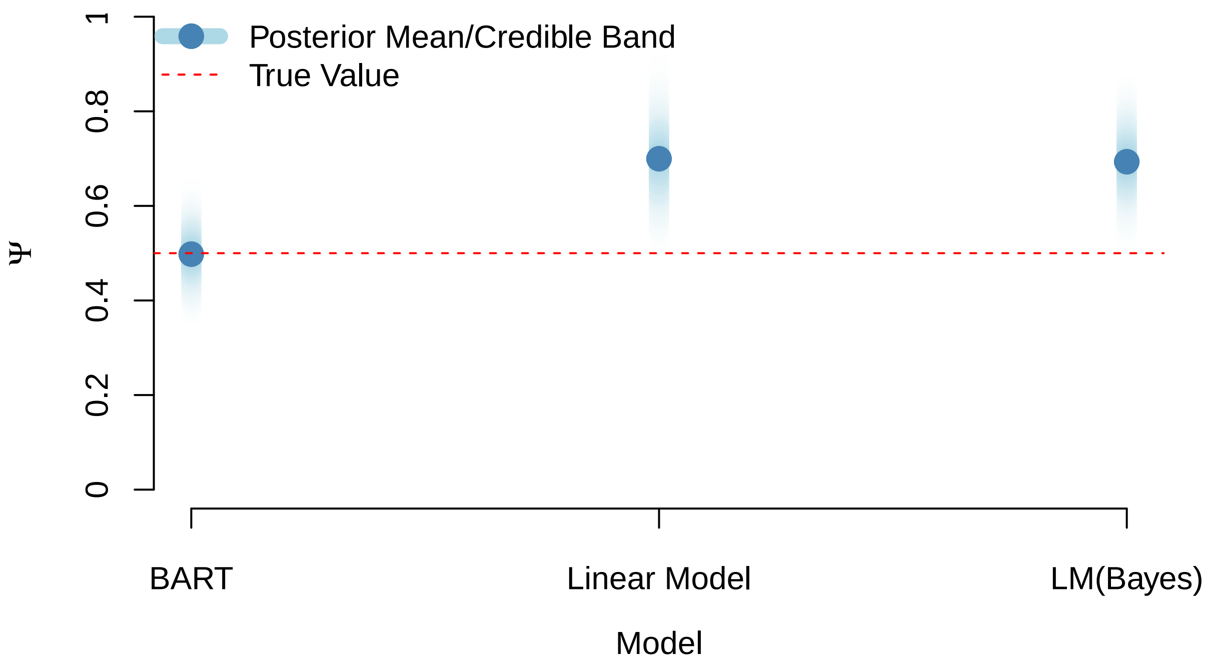
As this is a simulation study, we can compute the true ATE:

$$ATE = E[E[Y \mid X = 1, Z] - E[Y \mid X = 0, Z]] = E[Z + 0.5Z^2] = 0.5$$

We can compare the results by plotting the posterior mean and 95% credible intervals for ATE derived by BART and the bootstrap mean and quantiles from the linear regression model:

```r
#### ----------------------- Plot Results       ----------------------------####
all_psi = cbind(psi_bart, psi_lm_boot, psi_bayeslm)

post_mean = colMeans(all_psi)


plot(post_mean, pch=20, col='blue', ylim= c(0, 1),
     ylab=TeX("$\\Psi$"), xlab=TeX("Model"), axes=F )

axis_labs = c('BART','Linear Model','LM(Bayes)')

axis(1, at = 1:3, labels = axis_labs, padj = .5 )
axis(2, at = seq(0,1, .2), labels = seq(0, 1, .2) )


### Plot posterior credible Intervals
colfunc <- colorRampPalette(c("white", "lightblue"))
ci_perc = seq(.99,.01,-.01)
# ci_perc = c(.95,.90,.80)
# ci_perc = c(.95)
colvec = colfunc(length(ci_perc))
names(colvec) = ci_perc

for(i in ci_perc){
  pci = apply(all_psi, 2, quantile, probs=c( (1-i)/2, (1+i)/2  ) )
  segments(1:4,pci[1,], 1:4, pci[2,], col=colvec[as.character(i)], lwd=10 )
}
###

points(post_mean, col='steelblue', pch=20, lwd=8)
abline(h=.5, col='red', lty=2)

legend('topleft',
       legend = c('Posterior Mean/Credible Band', 'True Value'),
       col = c('lightblue', 'red'), pch=c(20,NA), lty=c(1, 2), lwd=c(8,1), bty='n')
```

```
legend('topleft',
       legend = c('', ''),
       col = c('steelblue', 'red'), pch=c(20,NA), lty=c(NA, 2), lwd=c(8,1), bty='n')
```



We can see the mis-specification of the linear regression model caused a large bias in estimating the ATE, while BART recovers the true value pretty well.

References

- The paper this lab is based on: Oganisian, A, Roy, JA. A practical introduction to Bayesian estimation of causal effects: Parametric and nonparametric approaches. Statistics in Medicine. 2021; 40: 518– 551. https://doi.org/10.1002/sim.8761
- For more details on BART, please refer to this: Tan, YV, Roy, J. Bayesian additive regression trees and the General BART model. Statistics in Medicine. 2019; 38: 5048– 5069. https://doi.org/10.1002/si m.8347