Simple Example

2022-11-23

Goal of this doc

To simulate a really simple case to try to understand the problem with increasing validation sizes with the Wang et al approach. I'm going to dramatically simplify things to try to make it easier to follow.

Load some libraries we will need

```
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(magrittr)
library(splines)
library(modelr)
library(broom)
```

```
##
## Attaching package: 'broom'
```

```
## The following object is masked from 'package:modelr':
##
##     bootstrap
```

```
library(tidyr)
```

```
##
## Attaching package: 'tidyr'
```

```
## The following object is masked from 'package:magrittr':
##
##     extract
```

```
library(gam)
```

```
## Loading required package: foreach
```

```
## Loaded gam 1.22
```

```
library(patchwork)
```

Simulate one very simple case

Let's remove all the bootstrapping and complicated data generation just so we can understand the global behavior of different operations. First I'll just simulate a really simple case:

```
## Set sample sizes with imbalance
n_train = 200
n_test = 200
n_valid = 10000
n_tot = n_train + n_test + n_valid

## Make some covariates
x1 = rnorm(n_tot)
x2 = rnorm(n_tot)
x3 = rnorm(n_tot)
err = rnorm(n_tot)

## Make an outcome

y = 2*x1 + 3*x2^2 + 0.5*x3^3 + err

## Create an indicator of training,testing,or validation

set_ind = c(rep("train",n_train),
            rep("test",n_test),
            rep("valid",n_valid))

## Build the data set

dat = data.frame(y,x1,x2,x3,set_ind)
```

Fit a prediction model

Ok now lets get the predictions - filter to the training

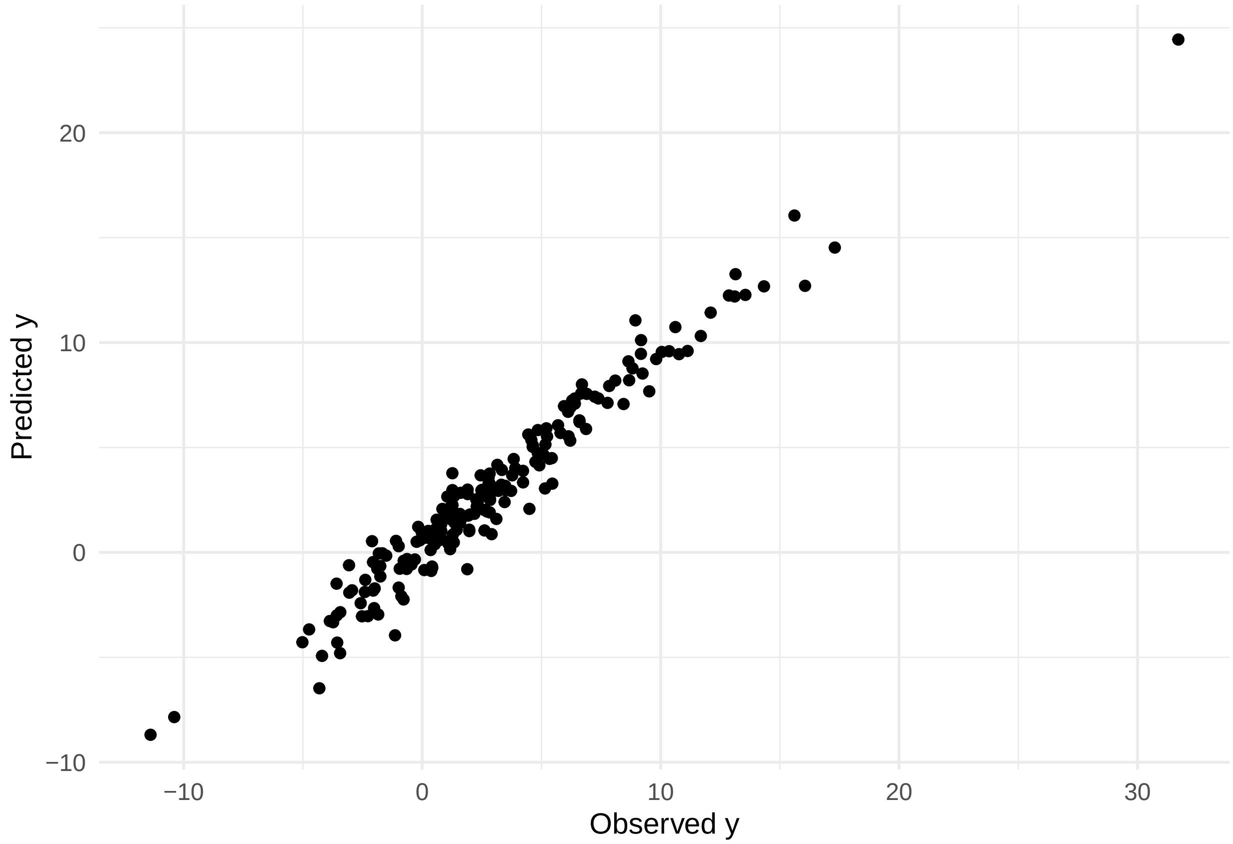
```
## Build the predictor and make the predictions

train_dat  = dat %>%
  filter(set_ind == "train") %>%
  select(y,x1,x2,x3)

pred_mod = gam(y ~ s(x1) + s(x2) + s(x3), data = train_dat)
```

Make the predictions for all the values (note we will only use the predictions in testing and validation data sets):

```
dat$pred = predict(pred_mod,dat)
```

Let's make sure the prediction looks good:

```
## Plot only on the test set
dat %>%
  filter(set_ind=="test") %>%
  ggplot(aes(x=y,y=pred)) +
  geom_point() + theme_minimal() +
  xlab("Observed y") +
  ylab("Predicted y")
```



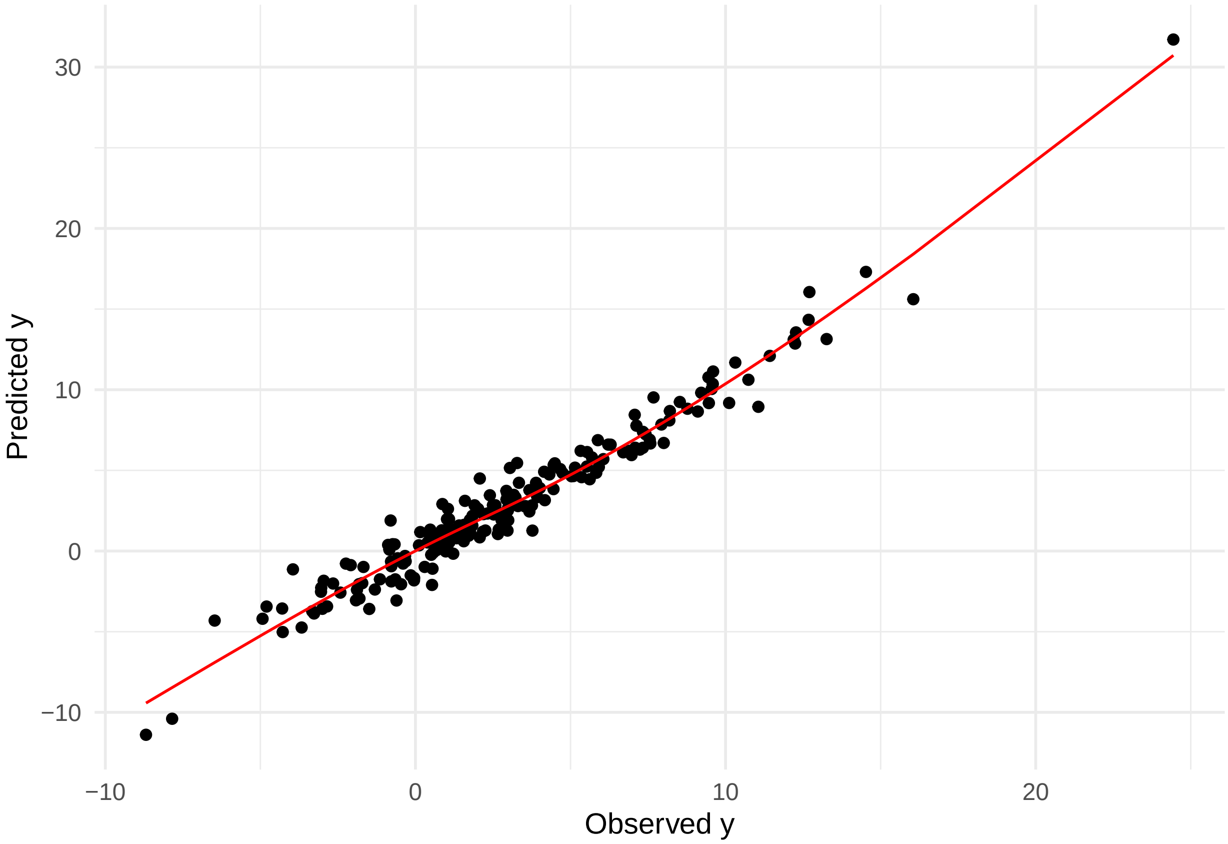
Let’s now do one bootstrap iteration of the Wang et al approach at a time

First we get the relationship model. Here we use a gam (like we do in Sara’s code) because we observe that the relationship between the predicted and observed y’s doesn’t look purely linear.

```
## Filter to the test set
test_dat = dat %>%
  filter(set_ind == "test")

## Fit the relationship model

rel_mod = lm(y ~ ns(pred,df=3),data=test_dat)


## Get the residual variance for use in simulating
sigma2 = sigma(rel_mod)^2
```

Let’s make sure the relationship model looks ok:

```
test_dat %>%
  add_predictions(rel_mod,var="fitted") %>%
  arrange(y) %>%
  ggplot(aes(x=pred,y=y)) +
  geom_point() + theme_minimal() +
  xlab("Observed y") +
  ylab("Predicted y") +
  geom_line(aes(x=pred,y=fitted),col="red")
```



Now let's do a step of the simulation for the validation set

I'm going to follow the Wang algorithm directly here.

Step 0. Subset to validation data

```
valid_dat = dat %>%
  filter(set_ind == "valid")
```

Step 1. Sample with replacement from x's

This may not be a strictly necessary step, but we do it in the boostrap iterations so doing this for completeness

```
sim_dat = valid_dat %>% sample_n(size=n_valid,replace=TRUE)
```

Step 2. Sample y's from the relationship model

This is where I could imagine something went wrong in Sara's code (I was having trouble understanding it myself) but basically we need to sample from the regression model for the predicted values:

```
y_sim = rnorm(n_valid,
              mean=sim_dat$pred,
              sd = sqrt(sigma2))
sim_dat = cbind(sim_dat,y_sim)
```

Step 3. Compute coefficients

Let's compare coefficients and sds from the different models

```
## Oracle
lm(y ~ x1,data=valid_dat) %>% tidy()
```

```
## # A tibble: 2 x 5
##   term        estimate std.error statistic p.value
##   <chr>          <dbl>     <dbl>     <dbl>   <dbl>
## 1 (Intercept)     2.96    0.0481      61.5       0
## 2 x1              1.96    0.0477      41.1       0
```

```
## Naive Post-PI
lm(pred ~ x1,data=valid_dat) %>% tidy()
```

```
## # A tibble: 2 x 5
##   term        estimate std.error statistic p.value
##   <chr>          <dbl>     <dbl>     <dbl>   <dbl>
## 1 (Intercept)     2.94    0.0426      69.1       0
## 2 x1              2.06    0.0423      48.6       0
```

```
## Post-PI via simulated data
lm(y_sim ~ x1,data=sim_dat) %>% tidy()
```

```
## # A tibble: 2 x 5
##   term        estimate std.error statistic p.value
##   <chr>          <dbl>     <dbl>     <dbl>   <dbl>
## 1 (Intercept)     2.89    0.0432      66.8       0
## 2 x1              2.01    0.0431      46.7       0
```

Let’s do the same thing for many simulations just to make sure we understand what’s going on here

```
## Set the number of replications
n_rep = 500

## Set sample sizes with imbalance
n_train = 1000
n_test = 1000
n_valid = 3000
n_tot = n_train + n_test + n_valid


beta_wang = beta_naive = beta_oracle = se_wang = se_naive = se_oracle = rep(NA,n_rep)

for(k in 1:n_rep){
  ## Make some covariates
  x1 = rnorm(n_tot)
  x2 = rnorm(n_tot)
  x3 = rnorm(n_tot)
  err = rnorm(n_tot,sd=4)

  ## Make an outcome
  y = 2*x1 + 3*x2^2 + 0.5*x3^3 + err

  ## Create an indicator of training,testing,or validation
  set_ind = c(rep("train",n_train),
            rep("test",n_test),
            rep("valid",n_valid))

  ## Build the data set
  dat = data.frame(y,x1,x2,x3,set_ind)


  ## Fit the model
  train_dat  = dat %>%
  filter(set_ind == "train") %>%
  select(y,x1,x2,x3)

  pred_mod = gam(y ~ s(x1) + s(x2) + s(x3), data = train_dat)

  ## Add the predictions
  dat$pred = predict(pred_mod,dat)

  ## Filter to the test set
  test_dat = dat %>%
    filter(set_ind == "test")

  ## Fit the relationship model
  rel_mod = lm(y ~ ns(pred,df=3),data=test_dat)

  ## Get the residual variance for use in simulating
  sigma2 = sigma(rel_mod)^2


  ## Filter to the validation set
  valid_dat = dat %>%
    filter(set_ind == "valid")

  ## Get the simulated data
  sim_dat = valid_dat %>% sample_n(size=n_valid,replace=TRUE)

  ## Sample simulated y's
  y_sim = rnorm(n_valid,
              mean=sim_dat$pred,
              sd = sqrt(sigma2))
  sim_dat = cbind(sim_dat,y_sim)

  ## Fit the models

  lm_oracle = lm(y ~ x1,data=valid_dat)
  lm_naive = lm(pred ~ x1,data = valid_dat)
  lm_wang = lm(y_sim ~ x1,data=sim_dat)

  ## Get the coefficients
  beta_oracle[k] = lm_oracle$coefficients[2]
  beta_naive[k] = lm_naive$coefficients[2]
  beta_wang[k] = lm_wang$coefficients[2]

  ## Get the parametric ses
  se_oracle[k] = summary(lm_oracle)$coefficients[2,2]
```

```
  se_naive[k] = summary(lm_naive)$coefficients[2,2]
  se_wang[k] = summary(lm_wang)$coefficients[2,2]

 cat(k)
}
```

```
## 1234567891011121314151617181920212223242526272829303132333435363738394041424344454647484950515253545
```

Let's look at what the coverage looks like

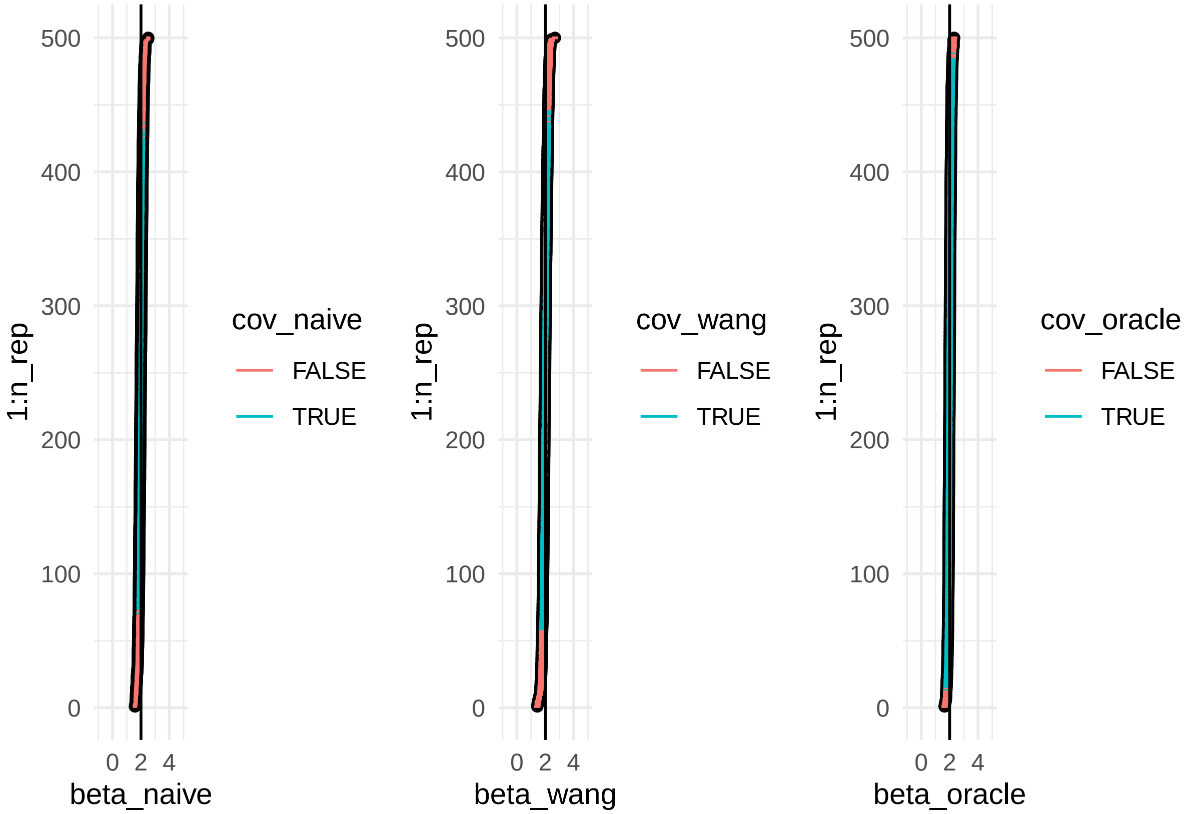
```
results = data.frame(beta_oracle,se_oracle,beta_wang,se_wang,beta_naive,se_naive)

results = results %>%
  mutate(up_oracle = beta_oracle + 1.96*se_oracle,
         low_oracle = beta_oracle - 1.96*se_oracle,
         cov_oracle = ((up_oracle > 2) & (low_oracle < 2)),
         ## Wang
         up_wang = beta_wang + 1.96*se_wang,
         low_wang = beta_wang - 1.96*se_wang,
         cov_wang = ((up_wang > 2) & (low_wang < 2)),

         ## Naive
         up_naive = beta_naive + 1.96*se_naive,
         low_naive = beta_naive - 1.96*se_naive,
         cov_naive = ((up_naive > 2) & (low_naive < 2)),
         )
```

Make some plots

Coverage is improved from naive, but not as good as oracle

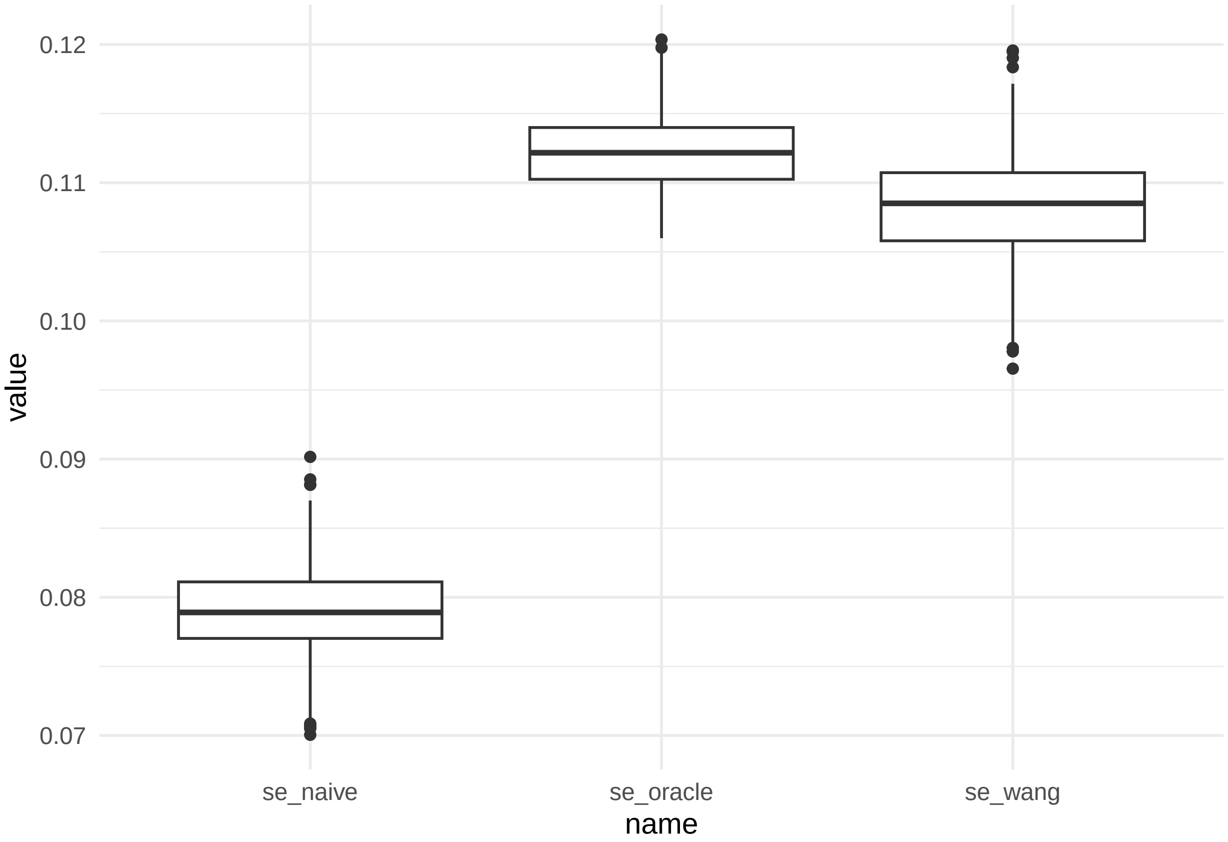
```
p1 = results %>%
  arrange(beta_naive) %>%
  ggplot(aes(y = 1:n_rep))  + geom_point(aes(x=beta_naive,y=1:n_rep)) +
  geom_linerange(aes(xmin=low_naive, xmax=up_naive,col=cov_naive)) +
  theme_minimal() +
  geom_vline(xintercept=2) +
  xlim(-1,5)

p2 =  results %>%
  arrange(beta_wang) %>%
  ggplot(aes(y = 1:n_rep)) + geom_point(aes(x=beta_wang,y=1:n_rep)) +
  geom_linerange(aes(xmin=low_wang, xmax=up_wang,col=cov_wang)) +
  theme_minimal() +
  geom_vline(xintercept=2) +
  xlim(-1,5)

p3 = results %>%
  arrange(beta_oracle) %>%
  ggplot(aes(y = 1:n_rep)) + geom_point(aes(x=beta_oracle,y=1:n_rep)) +
  geom_linerange(aes(xmin=low_oracle, xmax=up_oracle,col=cov_oracle)) +
  geom_vline(xintercept=2) +
  theme_minimal() +
  xlim(-1,5)
```

```
p1 + p2 + p3
```



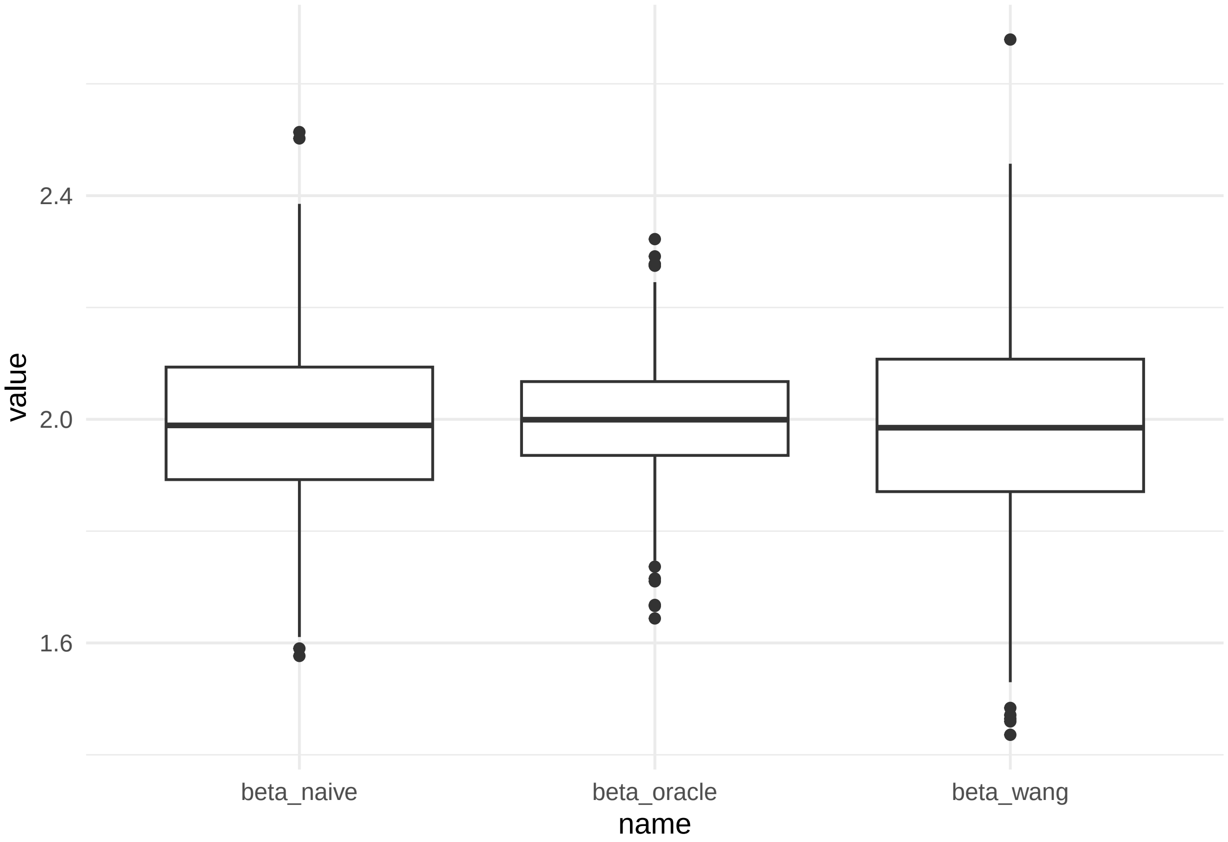
Standard errors are closer

```
results %>%
  select(se_naive,se_wang,se_oracle) %>%
  pivot_longer(1:3) %>%
  ggplot(aes(x=name,y=value)) +
  geom_boxplot() +
  theme_minimal()
```



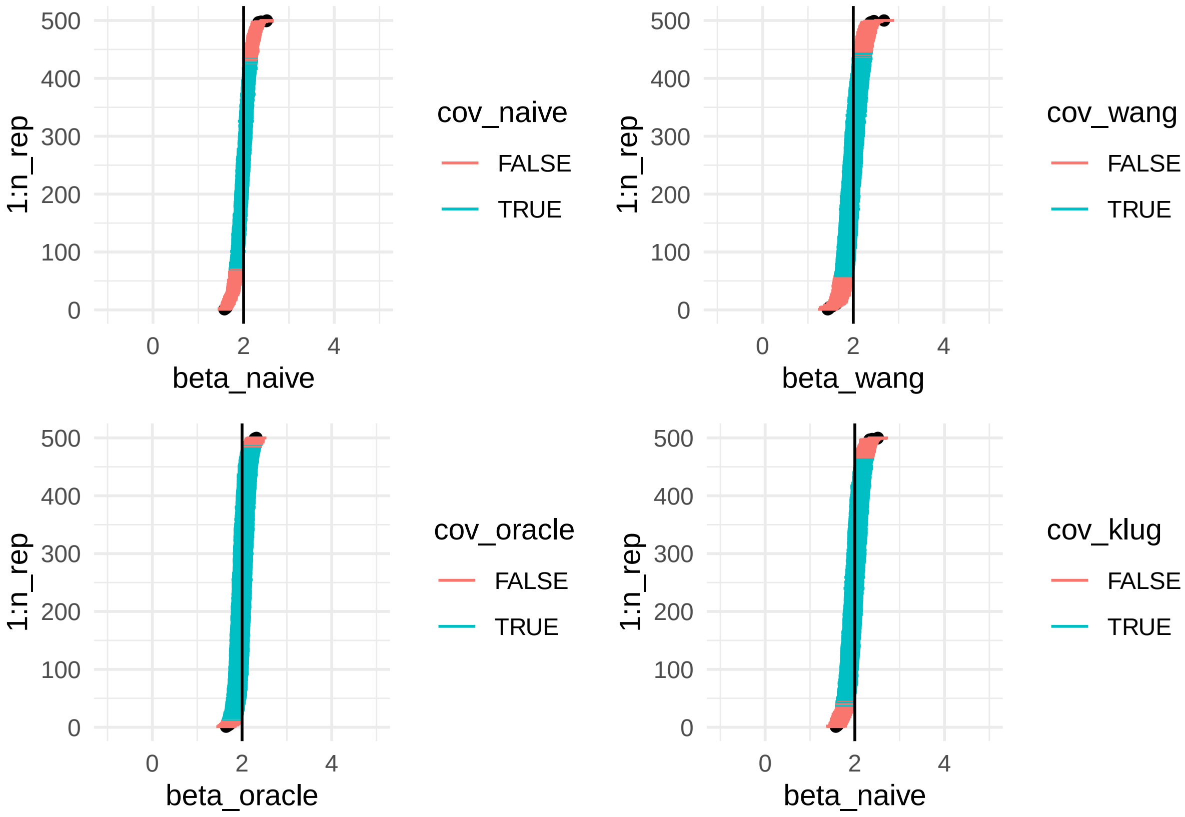
There are too many "big" values of the estimate

```
results %>%
  select(beta_naive,beta_wang,beta_oracle) %>%
  pivot_longer(1:3) %>%
  ggplot(aes(x=name,y=value)) +
  geom_boxplot() +
  theme_minimal()
```



What if we just replace the estimate with the naive estimate, but use the Wang SE (klugey version of what the bootstrap would do)

```
results = results %>%
  mutate(
         up_klug = beta_naive + 1.96*se_wang,
         low_klug = beta_naive - 1.96*se_wang,
         cov_klug = ((up_klug > 2) & (low_klug < 2)),
         )

p4 = results %>%
  arrange(beta_naive) %>%
  ggplot(aes(y = 1:n_rep)) + geom_point(aes(x=beta_naive,y=1:n_rep)) +
  geom_linerange(aes(xmin=low_klug, xmax=up_klug,col=cov_klug)) +
  geom_vline(xintercept=2) +
  theme_minimal() +
  xlim(-1,5)


(p1 + p2)/(p3 + p4)
```



Look at coverage

Looks like the klug works in this one random case :).

```
results %>%
  summarize(cov_naive = mean(cov_naive),
            cov_wang = mean(cov_wang),
            cov_klug = mean(cov_klug),
            cov_oracle = mean(cov_oracle))
```

```
##   cov_naive cov_wang cov_klug cov_oracle
## 1     0.718    0.772    0.848      0.944
```