STAT 2270 Homework 2 Fall 2020

Manuel Alejandro Garcia Acosta

9/29/2020

NOTES:

- As usual, I'll be setting a seed whenever I run a function which uses random variables for reproducibility purposes.
- For many codes that take quite a while to compile I saved the R objects into RData files, I include these in my submission. I load these files at the beggining of each exercise when compiling.
- Related to the above, codes that weren't run for compiling will have the eval=FALSE option enabled. Need to change this to eval=TRUE if you want to run them.

Exercise 6

Bootstrapping can be used to estimate the sampling distribution of most statistics, but not all:

Part (a)

Give an example of a statistic for which the bootstrap struggles to provide an accurate sampling distribution.

Given a sample $X_1, ..., X_n$. Consider

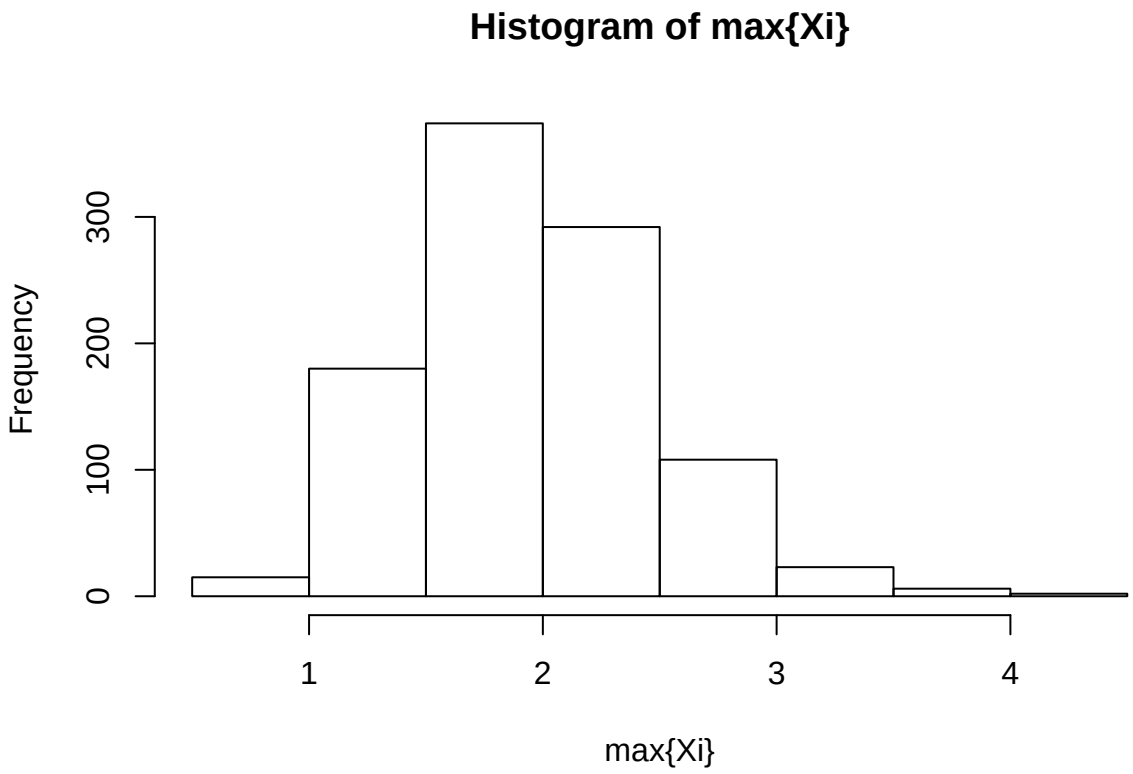
$$T(\mathcal{X}) = \max_{i \in \{1,...,n\}} \{X_i\}$$

Part (b)

Demonstrate your example from part (a). Take 1000 samples of size 25 from a standard normal distribution, calculate your statistic on each one, and create a histogram.

Here I just create 1000 samples with 25 observations each from a standard normal distribution and plot an histogram of the statistic $max\{X_i\}$.

```
set.seed(2019)
num <- 1:1000
norm25 <- function(x){
  rnorm(25)
}
samples <- lapply(num, norm25)

vect.max <- sapply(samples,max)
```



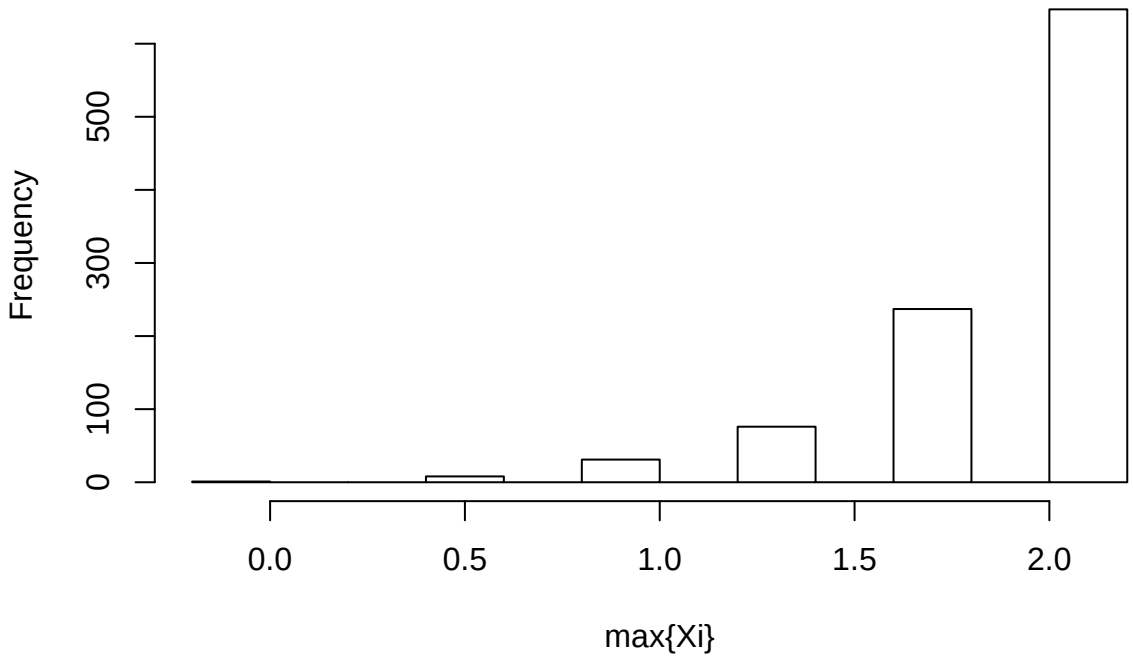
This will be considered as the true distribution of $T(\mathcal{X})$.

Part (c)

Now select one of your 1000 samples uniformly at random and take 1000 bootstrap samples. With each bootstrap sample, calculate the statistic and create a histogram.

Here the 647th sample was selected. I get 1000 bootstrap samples of such sample and then plot an histogram of $\max\{X_i\}$.

```
set.seed(2020)
ind <- sample(num, size = 1)
sample.ind <- samples[[ind]]
get.sample <- function(x){
  sample(sample.ind, size = 25, replace = TRUE)
}
boot.samples <- lapply(1:1000, get.sample)
boot.vect.max <- sapply(boot.samples,max)
```



This will be considered as the distribution of $T(\mathcal{X})$ according to bootstrapping sample 647.

Part (d)

How does your bootstrap distribution differ from the "true" sampling distribution? Give a brief explanation for why the bootstrap fails in your case.

We can notice right away that the values for $\max\{X_i\}$ in the bootstrap samples are smaller than in the "true" distribution. Moreover, we can see that the number of times $T(\mathcal{X}) = 1.11$ is higher than we would expect. In this case the bootstrap failed to *capture* the true distribution of $T(\mathcal{X})$ since at the moment sample 647 was chosen the distribution of the max was set to be biased to the left (because of such sample having observations with small values).

Exercise 7

Consider the function

$$f(x) = (x-2)^4 - 4(x-2)^3 + 5(x-2)$$

NOTE: You can find the objects of the code I'm not running while compiling the pdf in the file 'ex7.RData'

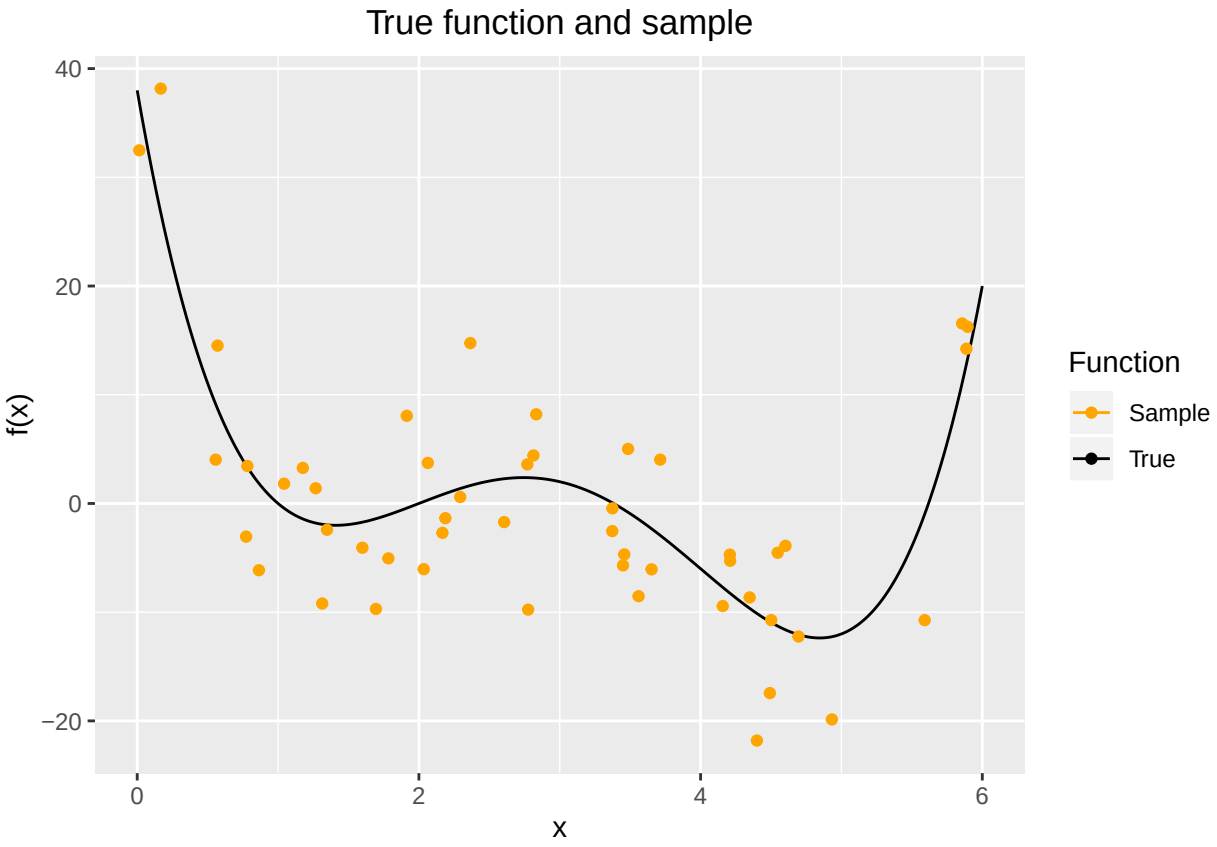
Part (a)

Plot this function (in the form of a line) in black for values of x from 0 to 6. Now consider the model $Y = f(X) + \epsilon$. Generate data by taking a random sample of size 50 from this model with $X_i \overset{iid}{\sim} U(0, 6)$ and $\epsilon_i \overset{iid}{\sim} N(0, 25)$ for $i = 1, ..., 50$ and add these points to your existing plot in a different color.

Here I just create the function and the observed sample.

```
function7 <- function(x){
  (x-2)^4 - 4*(x-2)^3 + 5*(x-2)
}

x <- seq(from = 0, to = 6, by = 0.01)
y <- sapply(x, function7)

df <- data.frame(x,y)
names(df) <- c('x','y')

set.seed(1981)
x.a <- runif(n = 50, min = 0, max = 6)
e.a <- rnorm(n = 50, mean = 0, sd = 5)
y.a <- sapply(x.a, function7) + e.a

data <- data.frame( 'x.a' = x.a, 'y.a' = y.a)
```

This is the plot of the original function along with the selected sample.



Part (b)

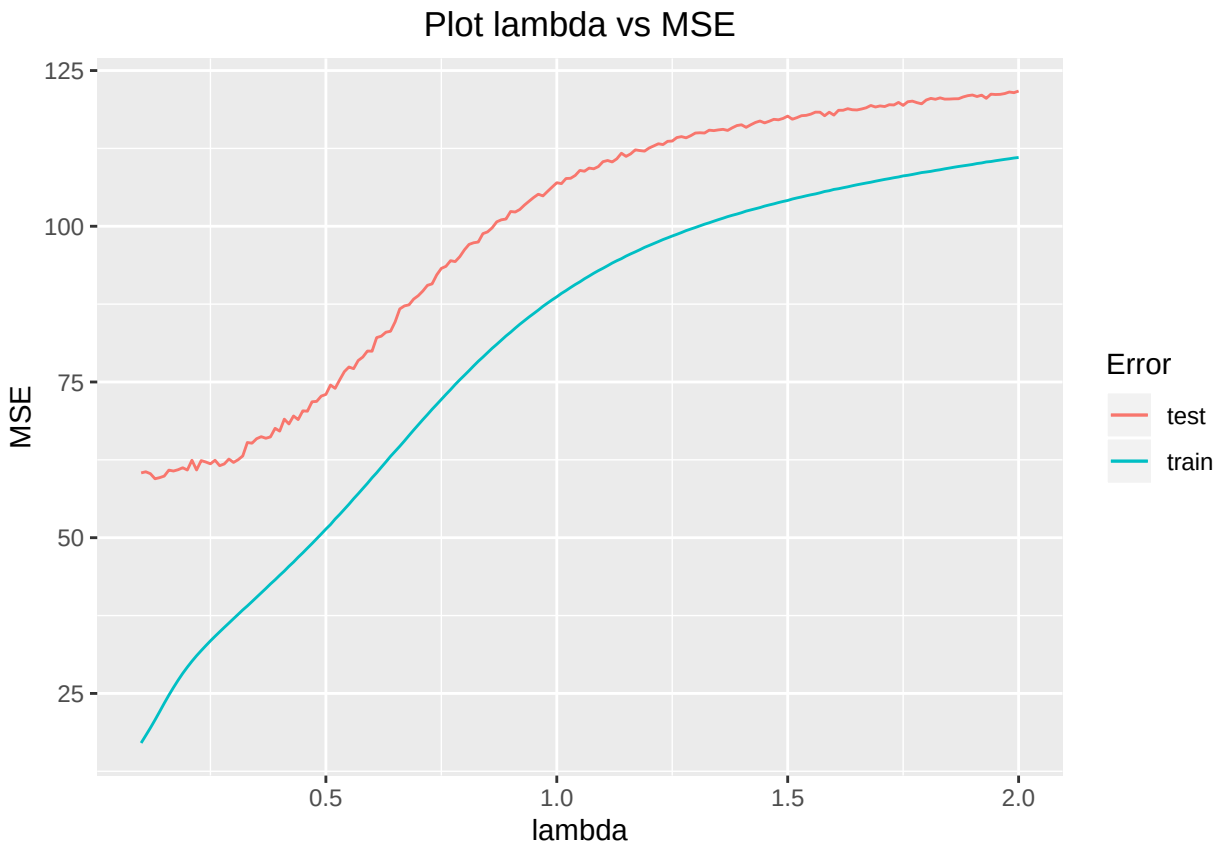
Consider doing kernel regression on the data from part (a) using a Gaussian kernel with bandwidth h. For an appropriate (wide enough) range of bandwidth values, plot the $10 - fold$ CV error of the model against h. Use this plot to select an appropriate value of h and call this h_G. Again plot the original function in black and now add this Gaussian kernel estimate to the plot (in the form of a line) in blue.

For part (b) I considered the set of lambdas $\lambda \in \{0.1, 0.11, 0.12, \ldots, 2\}$. For the Gaussian kernel I ran cross validation 200 times for each value of h and then reported the mean of the test MSE.

```
band <- seq(from = 0.1, to = 2, by =0.01)

set.seed(1970)
errors <- lapply(band, cv.kernel,x = x.a , y = y.a, kernel = 'gaussian', k = 10,
                 repeats = 200)

m.prueba <- do.call(rbind,errors)
m.prueba <- cbind(band,m.prueba)
m.prueba <- as.data.frame(m.prueba)
```

Here I just plot the train and test errors. After running cross validation the best value for h is $h_G = 0.13$ with corresponding test error $MSE = 59.46374$.

```
line.gaus <- sapply(x, nadaraya_watson, x= x.a, y = y.a, h=0.13,
                    kernel = 'gaussian' )
df.b <- data.frame(df,line.gaus)
```



Now I add the line estimate using the Gaussian kernel to the plot of part (a).

Part (c)

Now suppose we switch to the Epanechnikov kernel. Using the value h_G you found in part (b), add the Epanechnikov kernel estimate to the plot as a dotted red line.

Here I just plotted the estimated function using the Epanechnikov kernel for the bandwidth $h_G = 0.13$. The estimate is plotted as a red dotted line with label 'epan1'.

```
line.epan1 <- sapply(x, nadaraya_watson, x= x.a, y = y.a, h=0.13,
                     kernel = 'epanechnikov' )
df.b$line.epan1 <- line.epan1
```

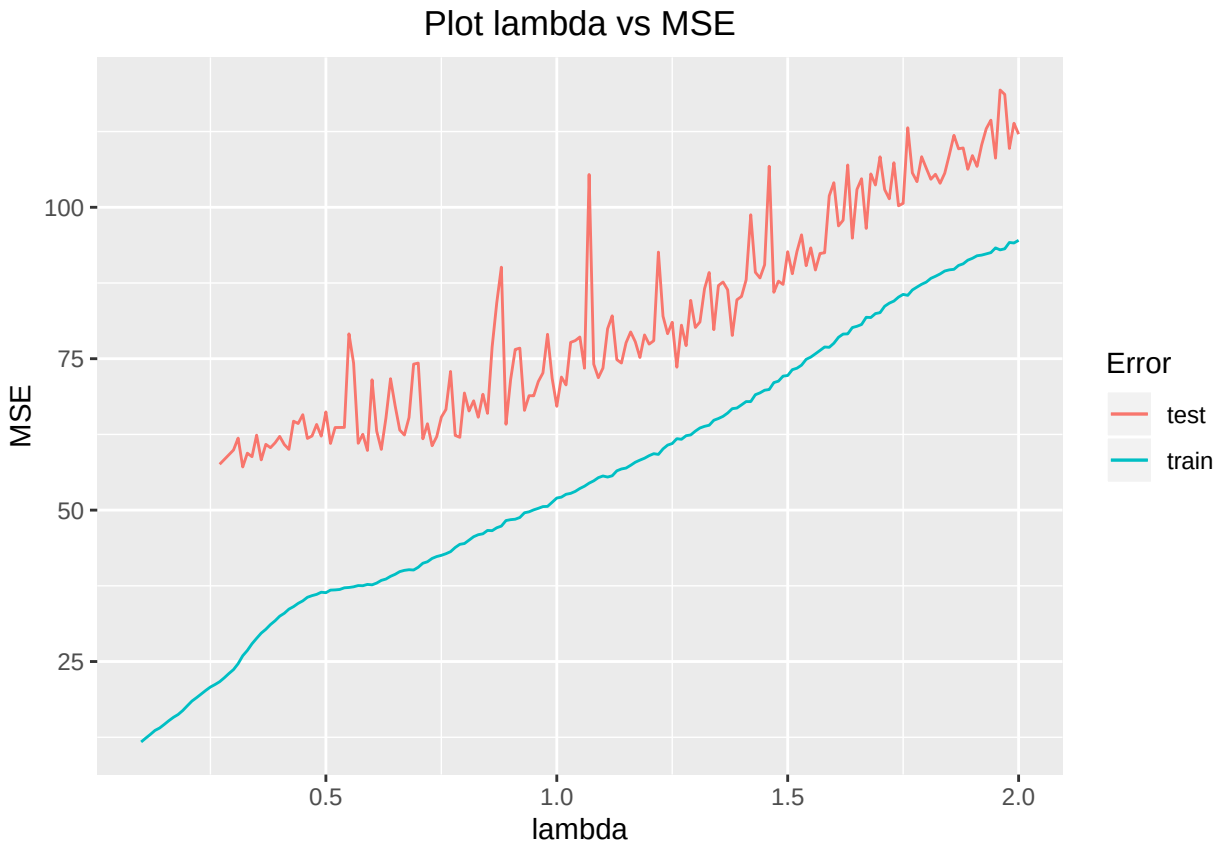
Part (d)

Now repeat part (b) using the Epanechnikov kernel and call the optimal bandwidth you find h_E. On the same plot from part (c), add the Epanechnikov kernel estimate with bandwidth h_E as a solid red line.

Here I ran kernel regression using cross validation. This time I just ran the procedure twice for each value of h because it was taking way more time to compute the estimates with the Epanechnikov kernel as compared to the Gaussian kernel.

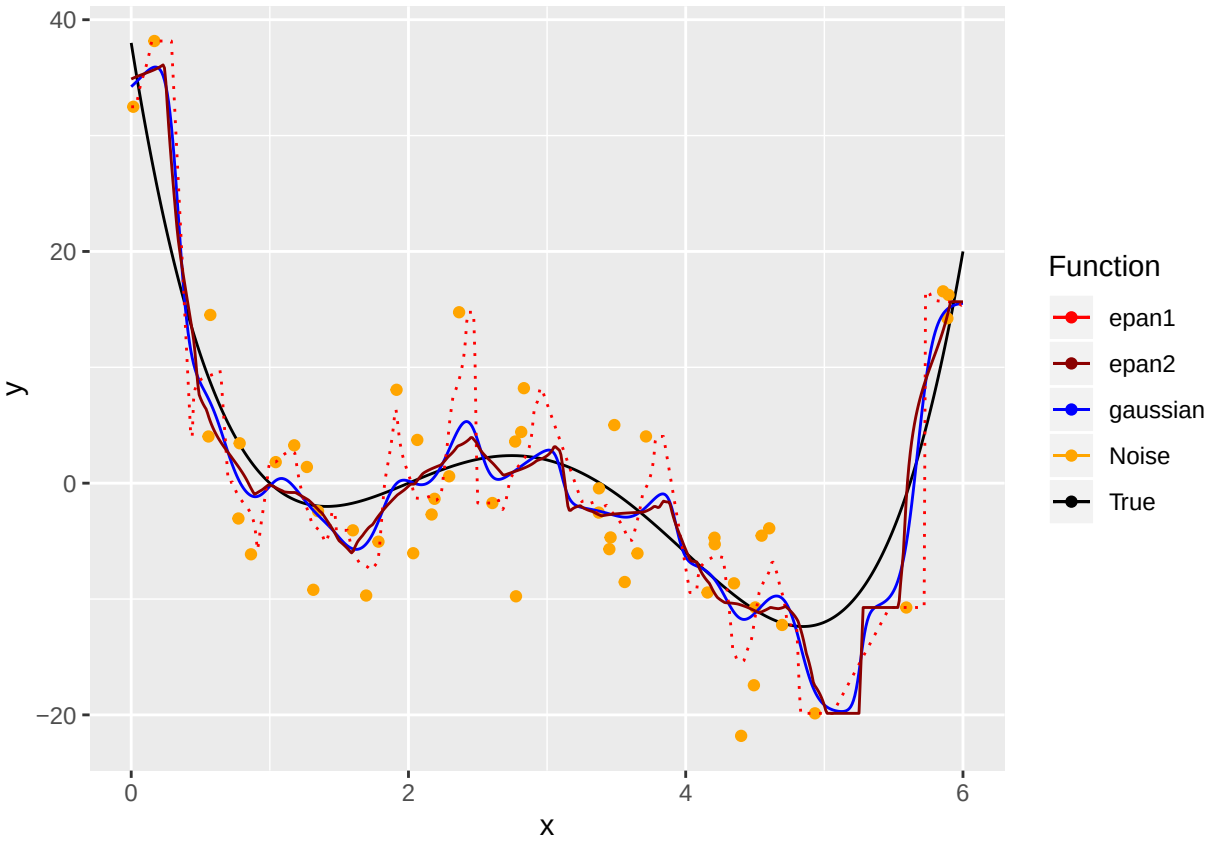
```
set.seed(1971)
errors1 <- lapply(band, cv.kernel,x = x.a , y = y.a, kernel = 'epanechnikov',
                  k = 10, repeats = 2)
m.prueba1 <- do.call(rbind,errors1)
m.prueba1 <- cbind(band,m.prueba1)
m.prueba1 <- as.data.frame(m.prueba1)
```

After running cross validation, the best bandwidth h for the Epanechnikov kernel was $h_E = 0.32$ with test error $MSE = 57.12501$.



Here I plotted the estimated function using the Epanechnikov kernel for the bandwidth $h_E = 0.32$. The estimate is plotted as a dark red line with label 'epan2'.

```
line.epan2 <- sapply(x, nadaraya_watson, x= x.a, y = y.a, h=0.32,
                     kernel = 'epanechnikov' )
df.b$line.epan2 <- line.epan2
```



Part (e)

Compare the two different fits of the Epanechnikov kernel regression estimate. How much did the bandwidth change in part (d)? How does the best Epanechnikov kernel estimate compare with the best Gaussian kernel estimate?

Even while running cross validation just twice for the Epanechnikov kernel and quite a lot more times for the Gaussian kernel the bandwidth didn't change much. Tuned values for the bandwidth went from $h_G = 0.13$ to $h_E = 0.32$, with a difference of 0.19. As we can see in the plot, the Epanechnikov estimate is really close to the Gaussian estimate when both have been tuned.

Exercise 8

Consider a standard linear model setup and suppose we have 50 (independent) covariates, but that only 5 of these (say the first five) have non-zero coefficients, with $\beta_i = i$ for $i = 1, ..., 5$. Finally, suppose we observe covariate values uniformly at random from the unit rectangle with standard iid Gaussian noise (i.e. $\epsilon \overset{iid}{\sim} N(0, 1)$).

NOTE: For this exercise I used instead $\beta_1 = 1$, $\beta_1 = 1$, $\beta_1 = 2$, $\beta_1 = 3$, $\beta_1 = 5$.

NOTE: You can find the objects of the code I'm not running while compiling the pdf in the file 'ex8.RData','ex8pte.RData' and 'ex8pte2.RData'.

Part (a)

Take a sample of size 25 and use 5-fold cross validation to fit a lasso model and record the resulting error. Call this $Err_{Lasso,1}(25)$.

First I created the dataset. The set of lambdas I used for part (a) was between 1 and 1/100.

As usual I set a seed before running functions which involve randomness.

```
iterations <- 25
variables <- 50
x <- matrix(ncol=variables, nrow=iterations, byrow = FALSE)
nom <- c()

set.seed(1991)
for(i in 1:variables){
  temp <- runif(25, min = 0, max = 1)
  x[,i] <- temp
  nom[i] <- paste0('x',i)
}

colnames(x) <- nom

x1 <- x[,1]
x2 <- x[,2]
x3 <- x[,3]
x4 <- x[,4]
x5 <- x[,5]
e <- rnorm(25, mean = 0, sd = 1)
y <- x1+x2+2*x3+3*x4+5*x5+e

data <- data.frame(x,y)

# Creating a set of possible values for lambda
lambdas <- 10^seq(0,-2,length.out=100)
```

Now I run 5-fold cross validation 10 times to find $Err_{Lasso,1}(25)$.

```
set.seed(1995)
fit <- cv.lasso(x=x, y=y, alpha = 1, lambda = lambdas, k=5,repeats = 10)
best.lambda <- best.lambda.test(fit)
best.mse.test <- best.mse.lasso(fit)
```

Here The best lambda was $\lambda = 0.1123324$ and the error test was $MSE = 1.688348$.

Part (b)

Repeat part (a) 1000 times to obtain $Err_{Lasso,1}(25), \ldots, Err_{Lasso,1000}(25)$. In addition, each time you fit a lasso model, determine which covariates are given non-zero coefficient estimates. Take these covariates and fit a standard linear model on only these. Record this error to obtain $Err_{OLS,1}(25), \ldots, Err_{OLS,1000}(25)$.

For computing the 1000 errors, I just ran CV once each time. The result of this procedure can be found in 'output1'.

```
set.seed(1989)
fit1 <- lapply(1:1000, function(z){
  mse.lasso.ols( x = x, y = y, data = data, alpha = 1,
```

```
                  lambda = lambdas, k =5, repeats = 1 )
})
output1 <- do.call(rbind,fit1)
```

Part (c)

In general, define

$$\overline{Err}_{Lasso}(n) = \frac{1}{1000}\sum_{i=1}^{1000} Err_{Lasso,i}(n)$$

and equivalently for the OLS errors. Calculate $\overline{Err}_{Lasso}(25)$ and $\overline{Err}_{OLS}(25)$ and take the difference.

Here we just need to use the result from part (b). Computing such means gives $\overline{Err}_{Lasso}(25) = 1.764167$ and $\overline{Err}_{OLS}(25) = 0.1193118$. The difference is

$$\overline{Err}_{Lasso}(25) - \overline{Err}_{OLS}(25) = 1.644856$$

```
err.lasso.25 <- mean(output1[,2])
err.ols.25 <- mean(output1[,3])
diff.25 <- err.lasso.25 - err.ols.25
```

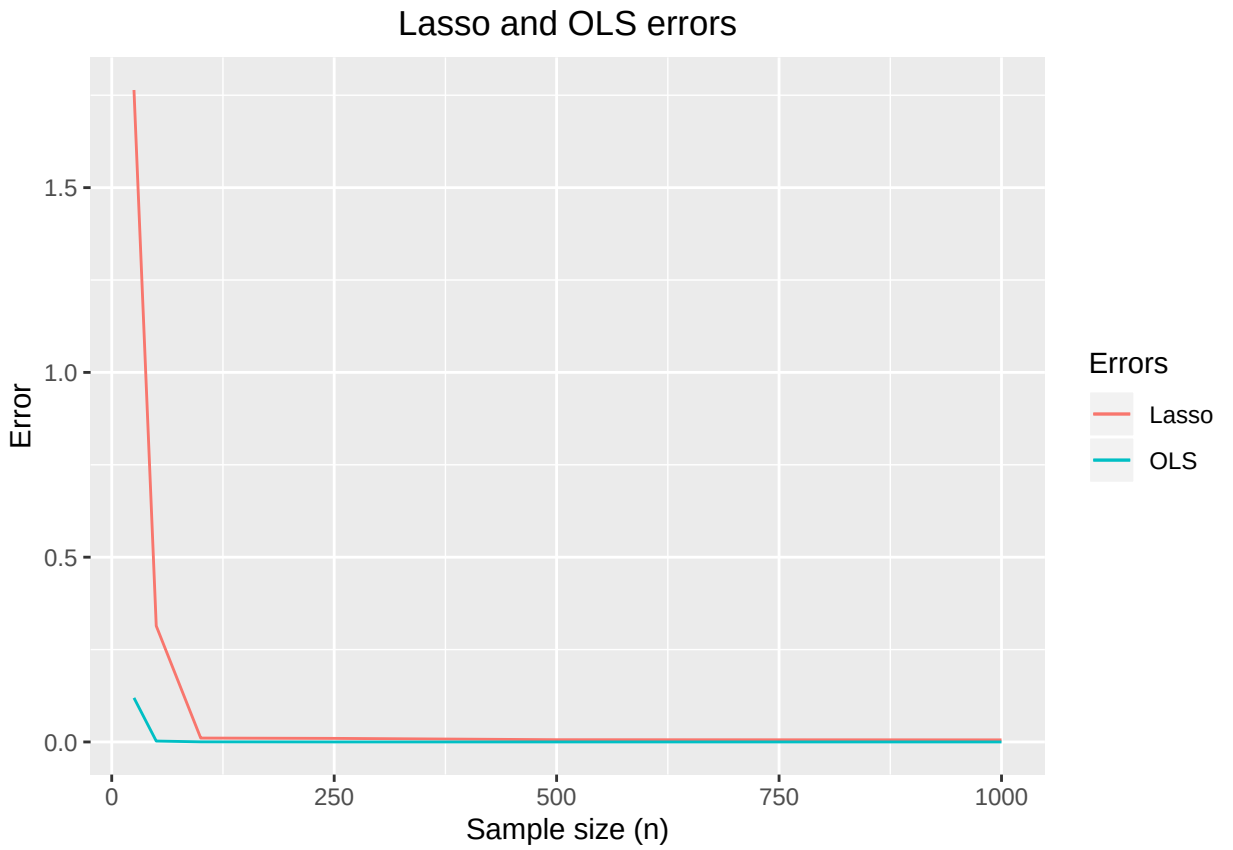
Part (d)

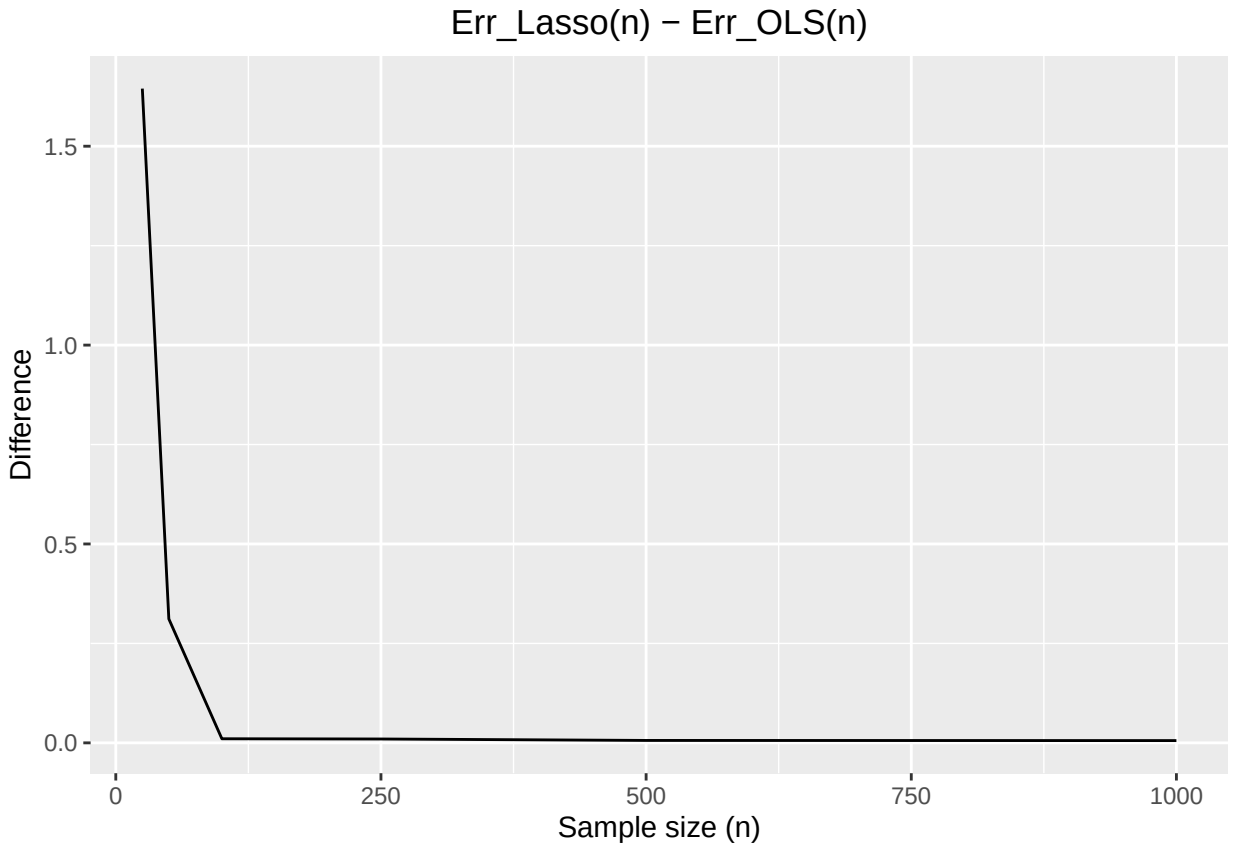
Explore if/how the difference in errors changes with n. For example, you might consider something like n = 25, 50, 100, 250, 500, 1000.

I got $\overline{Err}_{Lasso}(n)$ and $\overline{Err}_{OLS}(n)$ for sample sizes $n = 50,\ 100,\ 250,\ 500, 1000$ in addition to the ones obtained in part (c). All these are stored in the objects 'output2', 'output3', 'output4', 'output5', 'output6'. These simulations started to last for about an hour so I stored them and set the code to eval=FALSE.

I'm omitting the code for this in the pdf but you can find it in the Rmd file of my HW.

As we can see in the following plots, as the sample size n increases, $\overline{Err}_{Lasso}(n)$ and $\overline{Err}_{OLS}(n)$ become really close. From sample size 250 and bigger the difference becomes negligible.


Lasso and OLS errors
Error
1.5
1.0
0.5
0.0
0
250
500
750
1000
Sample size (n)
Errors
Lasso
OLS


Err_Lasso(n) − Err_OLS(n)
Difference
1.5
1.0
0.5
0.0
0
250
500
750
1000
Sample size (n)

Part (e)

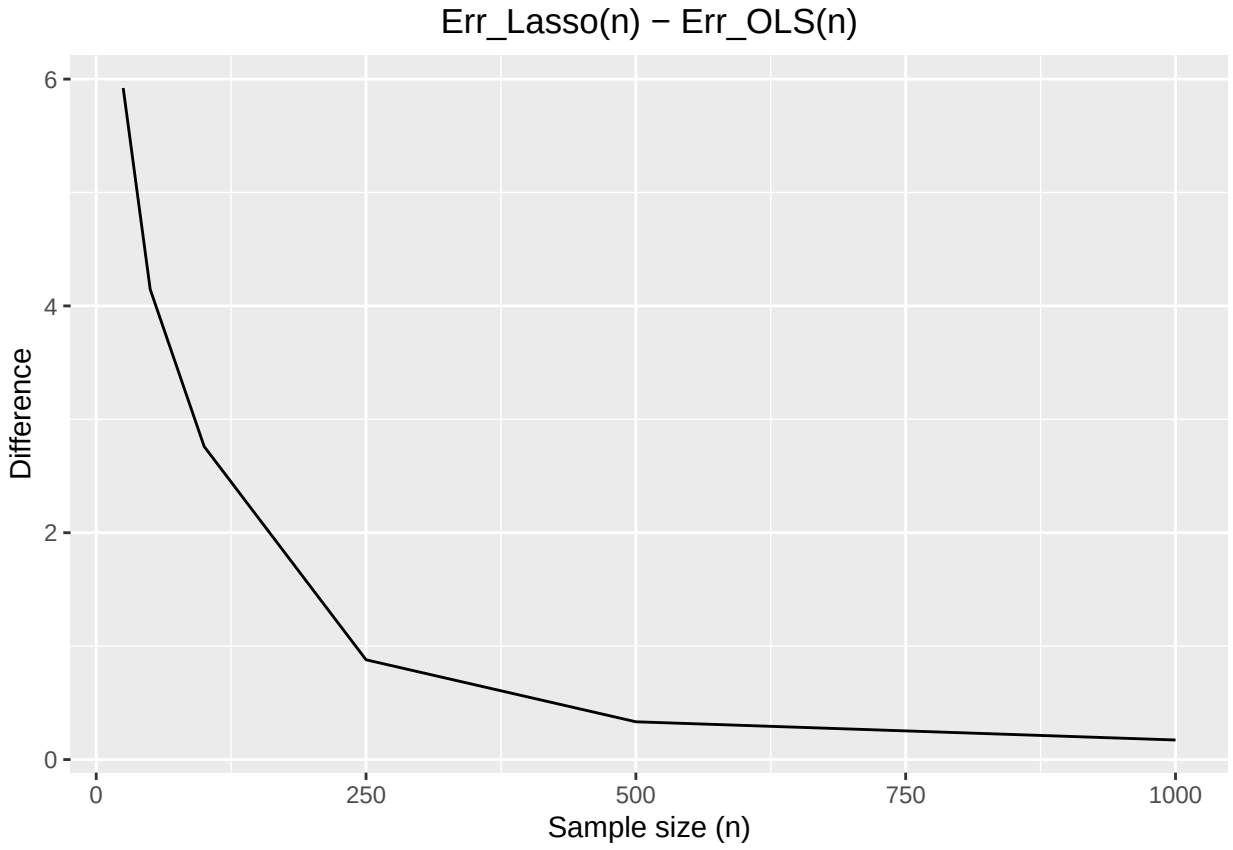
You might expect things to behave differently depending on the problem set-up. What if there were even more parameters and many of them had non-zero coefficients? What if the covariates weren't independent but (at least some) exhibited some correlation? Make some modifications to the original problem to explore these aspects and see what you can conclude.

Part (e)-1

What I did different here was that the I created 70 independent covariates and the first 60 had non-zero coefficients in the true model. In fact, $\beta_i = 1$ for $i = 1, ..., 60$. In addition, I computed the mean over 100 repetitions instead of 1000.

$$\overline{Err}_{Lasso}(n) = \sum_{i=1}^{100} Err_{Lasso,i}(n)$$

Similarly for $\overline{Err}_{OLS}(n)$. Sample sizes were once again $n = 25, 50, 100, 250, 500, 1000$. The result I got was quite similar, since as the sample size n increases, $\overline{Err}_{Lasso}(n)$ and $\overline{Err}_{OLS}(n)$ become really close as you can see in the plots.



Part (e)-2

What I did different here was that the I created 50 covariates and the covariates 16 through 30 were defined as

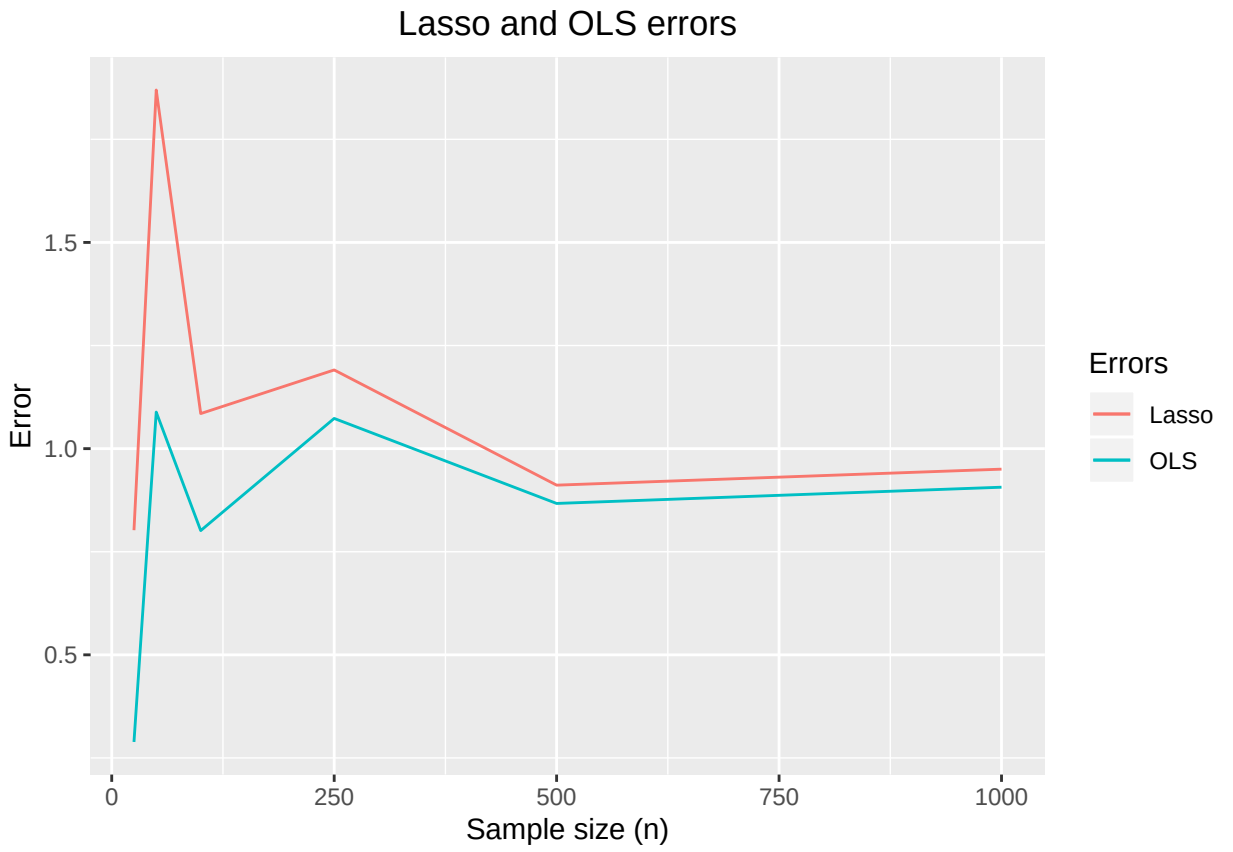
$$X_{i+15} = \sum_{j=1}^{i+1} X_j$$

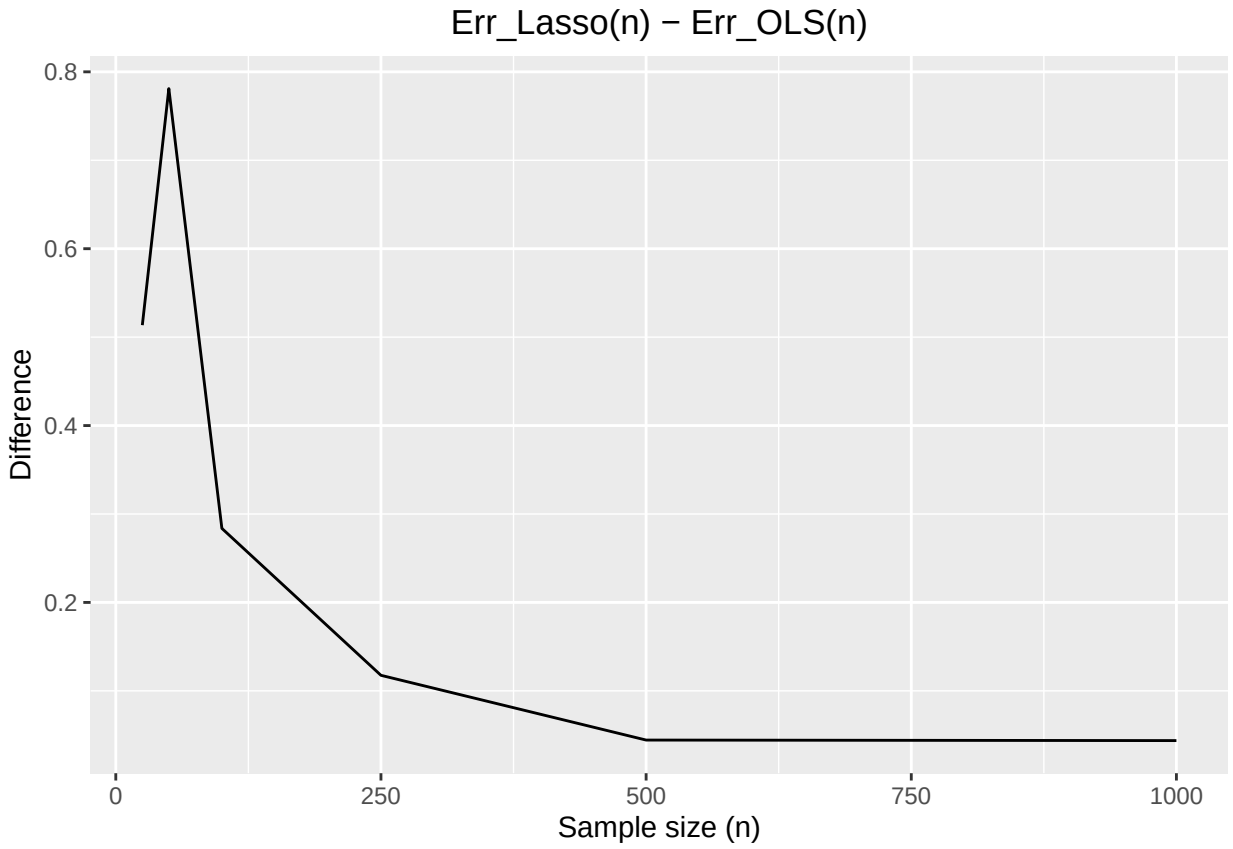
this way some of the covariates were correlated. In this dataset $X_{26}, X_{27}, X_{28}, X_{29}, X_{30}$ had non-zero coefficients in the true model ($\beta_i = 1$ for $i = 26, ..., 30$). In addition, I computed the mean over 20 repetitions instead of 1000.

$$\overline{Err}_{Lasso}(n) = \sum_{i=1}^{20} Err_{Lasso,i}(n)$$

Similarly for $\overline{Err}_{OLS}(n)$. Sample sizes were $n = 25, 50, 100, 250, 500, 1000$. The result I got was again similar. As the sample size n increases, $\overline{Err}_{Lasso}(n)$ and $\overline{Err}_{OLS}(n)$ become close.

It is worth mentioning that in the examples given in part(e)-1 and part(e)-2 the Lasso error doesn't approximate the OLS error as fast as it does in the original setup given on parts (a)-(d).


Lasso and OLS errors
Error
1.5
1.0
0.5
0
250
500
750
1000
Sample size (n)
Errors
Lasso
OLS


Err_Lasso(n) − Err_OLS(n)
Difference
0.8
0.6
0.4
0.2
0
250
500
750
1000
Sample size (n)

Part (f)

When (if ever) would it be good to do OLS after Lasso? Intuitively, why might it make sense to do that? This is closely related to the idea of the relaxed lasso [Meinshausen (2007), 'Relaxed lasso', Computational Statistics & Data Analysis]. Have a look at this paper and give a short description of what is being suggested. Code up this estimator and apply it to some of the settings above to see when it tends to outperform the Lasso and OLS methods alone.

The lasso shrinkage causes the estimates of the non-zero coefficients to be biased towards zero and in general they are not consistent. One way we can reduce this bias is running Lasso first to identidy the predictos with non-zero coefficient estimates (kind of doing model selection) first. Then we can run OLS on the remaining predictors with the benefit that these estimates are the best unbiased linear estimates.

Relaxed Lasso is proposed to deal with high dimensional data in situations were the number of predictor variables p is large and potentially larger than the number of observations n. Meinshausen(2007) states that for no value of γ the Bridge estimators

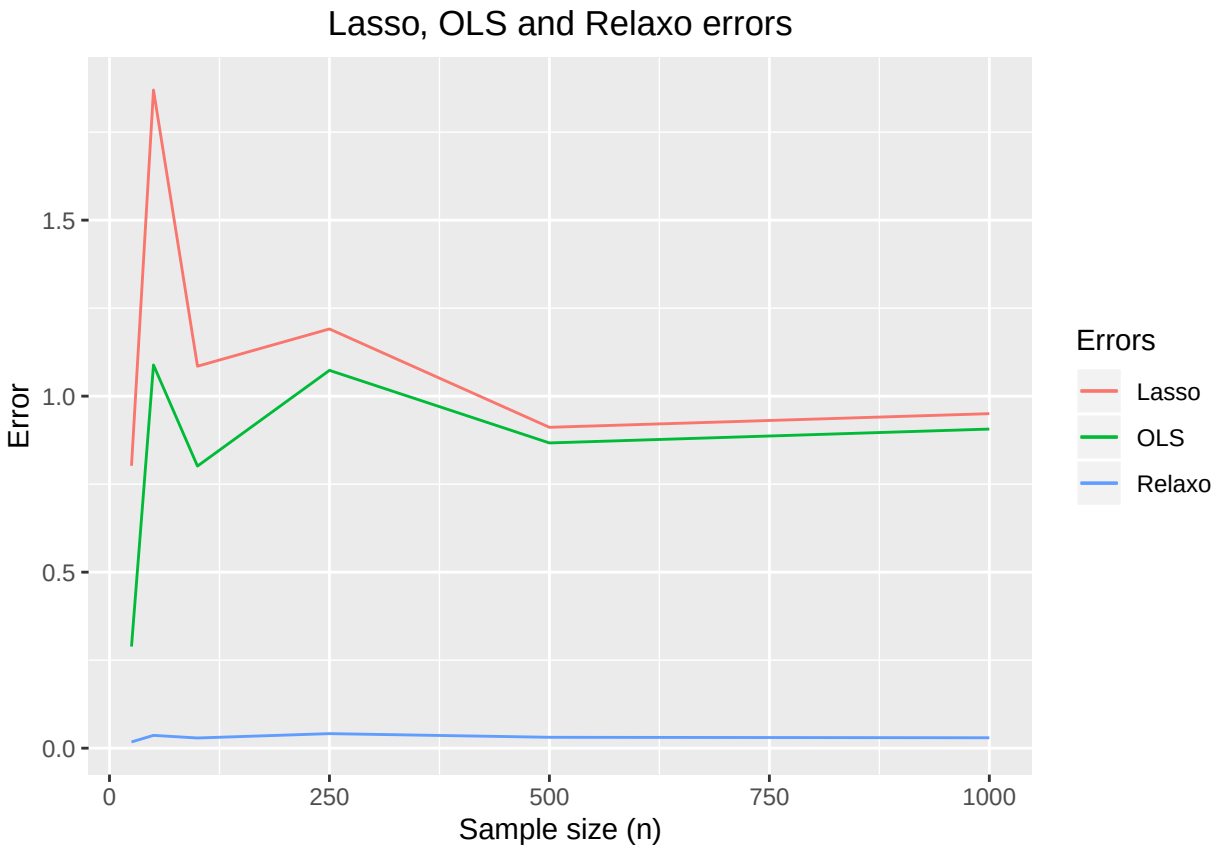
$$\hat{\beta}^{\lambda.\gamma} = \underset{\beta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} (Y_i - X_i^T \beta)^2 + \lambda ||\beta||_\gamma$$

achieve a satisfactory performance in terms of computational complexity and fast convergence rates (of the coefficients).

Meinshausen shows in his paper that the shrinkage of the Lasso leads to low convergence rate of the ℓ_2-loss for high dimensional problems where the number ofparameters p_n grows almost exponentially fast with n, so that $p_n >> n$. A two-stage procedure named *relaxed lasso* is proposed to work around this problem.

The idea is to use cross-validation to estimate the initial penalty parameter for the Lasso, then a second time to the predictors with non-zero coefficient estimates. "Since the variables in the second step have less "competition" from noise variables, cross-validation will tend to pick a smaller value for λ [the penalty parameter], and hence their coefficients will be shrunken less than those in the initial estimate" (Hastie, 2017, p. 91).

Here I applied Relazed Lasso, or *Relaxo*, to the setup from part(e)-2. The results were interesting since Relaxo outperformed OLS and $Lasso$.



Exercise 9

Assume the same original setup as in problem 8: a standard linear model with 50 independent covariates with non-zero coefficients for the first 5 variables only.

Part (a)

Draw a sample of size 25 with iid Gaussian noise assuming covariate values are sampled uniformly at random from the unit rectangle. Fit a lasso model using 5-fold CV to choose λ and fix this value of the tuning parameter.

I'll use the same setup as in exercise 8 part (a). Running 5-fold cross validation 20 times the best lambda is $\lambda = 0.129155$. I'll fix this parameter from now on.

```
iterations <- 25
variables <- 50
x <- matrix(ncol=variables, nrow=iterations, byrow = FALSE)
nom <- c()

set.seed(1991)
for(i in 1:variables){
  temp <- runif(25, min = 0, max = 1)
  x[,i] <- temp
  nom[i] <- paste0('x',i)
}
```

```r
colnames(x) <- nom

x1 <- x[,1]
x2 <- x[,2]
x3 <- x[,3]
x4 <- x[,4]
x5 <- x[,5]
e <- rnorm(25, mean = 0, sd = 1)
y <- x1+2*x2+3*x3+4*x4+5*x5+e

data <- data.frame(x,y)

# Creating a set of possible values for lambda
lambdas <- 10^seq(0,-2,length.out=100)
```

```r
set.seed(1995)
fit <- cv.lasso(x=x, y=y, alpha = 1, lambda = lambdas, k=5,repeats = 20)
best.lambda <- best.lambda.test(fit)
best.mse.test <- best.mse.lasso(fit)
```

Part (b)

Suppose we want to estimate the degrees of freedom for this model (i.e. we didn't know the theoretical result we talked about in class). We could bootstrap the rows of our data so that each time we get dataset of the form $\tilde{\mathcal{D}} = \{(\tilde{x_1}, \tilde{y_1}), ..., (\tilde{x_{25}}, \tilde{y_{25}})\}$ where we save the values $\tilde{y}_i$ as well as the fitted values $\hat{y}_i$ for $i = 1, ..., 25$. Repeat this process with B bootstrap samples and at the end, calculate the empirical covariance between each of the $\tilde{y}_i$ and the $\hat{y}_i$ which we can use as a proxy for $Cov(\hat{y}_i, y_i)$

$$\widehat{Cov}(\hat{y}_i, y_i) = \frac{1}{B}\sum_{b=1}^{B}\left(\hat{y}_i^{(b)} - \hat{\mathbb{E}}(\hat{y})\right)\left(\tilde{y}_i^{(b)} - \hat{\mathbb{E}}(\tilde{y})\right)$$

We could then estimate the degrees of freedom by

$$\hat{df}(\hat{y}) = \frac{1}{\sigma^2}\sum_{i=1}^{n}\widehat{Cov}(\hat{y}_i, y_i)$$

Using your data and λ from part (a), try this with $B = 1000$.

After doing the computations using the value for lambda obtained in part (a) I obtained $\hat{df}(\hat{y}) \approx 8$. It's wort mentioning that while running Lasso with the best lambda, i.e., $\lambda = 0.129155$ 13 of the predictor variables have non-zero coefficient estimates. The intercept has also a non-zero coefficient estimate.

```r
set.seed(1920)
allboots <- lapply(1:1000, function(z){ boots.fitted.tilde(data = data, lambda = best.lambda, alpha = 1
```

```r
cov.yi <- sapply(1:25, boots.cov.yi, boots.list =allboots)

exp.cov <- mean(cov.yi)
```

Part (c)

The estimator from (b) often doesn't work that well. Why not? Hint: It may not be what you first think ˘ consider what's fixed vs. random.

When we take the bootstrap samples $\mathcal{D}^{(\lfloor)} = \{(x_1^{(b)}, y_1^{(b)}), ..., (x_{25}^{(b)}, y_{25}^{(b)})\}$ we are randomizing pretty much everything with exception of the tuning parameter λ.

Part (d)

A better way to estimate the degrees of freedom is with a residual bootstrap. This is carried out in exactly the same fashion as part (b) except that now each bootstrap dataset will be of the form $\tilde{\mathcal{D}} = \{(x_1, \tilde{y}_1), ..., (x_{25}, \tilde{y}_{25})\}$ where

$$\tilde{y}_i = \hat{y}_i + \epsilon^*$$

ϵ^* selected uniformly at random from the 25 residuals calculated on the original data. Do this procedure to estimate the degrees of freedom with $B = 1000$. Why would you expect this to be a better estimate?

After doing the computations using the value for lambda obtained in part (a) I obtained $\hat{df}(\hat{y}) \approx 7$. As you might notice, this provide a better estimate for the degrees of freedom of the true model since Lasso for the true model has 6 degrees of freedom.

In part (d) we're fizing the covariates, and only the response (through the sampled residuals) is gonna be random. This results in a closer approximation of the actual *df* of the true model (6).

```
set.seed(1910)
original.fit <- glmnet(x,y, alpha =1, lambda = best.lambda)
best.coefficients <- predict(original.fit,s = best.lambda, newx = x,
                             type = 'coefficients')
origin.pred <- as.numeric(predict(original.fit, s = best.lambda, newx = x))
origin.resid <- y - as.numeric(origin.pred)

sample.boots <- lapply(1:1000, function(z){
  boots.sample.errors(x = x, fitted.y = origin.pred,
                      original.residuals = origin.resid)})

allboots.d <- lapply(sample.boots, boots.fitted.tilde.errors,
                     lambda = best.lambda, alpha = 1)

cov.yi.d <- sapply(1:25, boots.cov.yi, boots.list =allboots.d)

exp.cov.d <- mean(cov.yi.d)
```