Week 6 - Midterm Assignment: A Simulation Project

Mohammad Hassanpour, MH57

6/28/2022

true

Introduction

Simulation Study 1: Significance of Regression

In this simulation study we will investigate the significance of regression test. We will simulate from two different models:

1. The **"significant"** model

$$Y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \epsilon_i$$

where $\epsilon_i \sim N(0, \sigma^2)$ and

- $\beta_0 = 3$,
- $\beta_1 = 1$,
- $\beta_2 = 1$,
- $\beta_3 = 1$.

2. The **"non-significant"** model

$$Y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \epsilon_i$$

where $\epsilon_i \sim N(0, \sigma^2)$ and

- $\beta_0 = 3$,
- $\beta_1 = 0$,
- $\beta_2 = 0$,
- $\beta_3 = 0$.

For both, we will consider a sample size of 25 and three possible levels of noise. That is, three values of σ.

- $n = 25$
    - $\sigma \in (1, 5, 10)$

  Use simulation to obtain an empirical distribution for each of the following values, for each of the three values of σ, for both models.

- The F **statistic** for the significance of regression test.
- The **p-value** for the significance of regression test
- R^2

For each model and σ combination, use 2000 simulations. For each simulation, fit a regression model of the same form used to perform the simulation.

Use the data found in `study_1.csv` (study_1.csv) for the values of the predictors. These should be kept constant for the entirety of this study. The `y` values in this data are a blank placeholder.

Done correctly, you will have simulated the `y` vector $2(models) \times 3(sigmas) \times 2000(sims) = 12000$ times.

Methods

Simulation size is set.

```
sim_size = 2000
```

The data is loaded.

```
data1 = read.csv("study_1.csv")
```

The sample size is set.

```
n = 25
```

Vectors are created to store the true beta values and the true sigma values.

```
#significant
B0 = 3
B1 = B2 = B3 = 1
#non-significant
b0 = 3
b1 = b2 = b3 = 0

s1 = 1
s2 = 5
s3 = 10
```

The seed is set for the study.

```
birthday = 19781020
set.seed(birthday)
```

A function is created that takes the predictors x_1, x_2, x_3, true coefficient values $\beta_0, \beta_1, , \beta_2, \beta_3$, and true standard deviations of errors $\sigma_1, \sigma_2, \sigma_3$ and returns a data frame with the predictor and predictions generated from the true model and the given, fixed, predictor values.

```r
x1 = data1[, 2]
x2 = data1[, 3]
x3 = data1[, 4]

#beta and sigma values were defined in a previous chunk
update.y = function(x1, x2, x3, beta0, beta1, beta2, beta3, sigma) {
  epsilon = rnorm(length(x1), mean = 0, sd = sigma)
  y = beta0 + beta1 * x1 + beta2 * x2 + beta3 * x3 + epsilon
  data.frame(y, x1, x2, x3)
}
```

The value of σ is constant for each of 3 simulations of 2000, significance of regression is tested and the F-statistic, p-value, and $\backslash\mathrm{R}^2$ are stored in their respective vectors. This means the vectors must be created before simulation.

```r
f1 = f2 = f3 = rep(0,sim_size)

F1 = F2 = F3 = rep(0,sim_size)

p1 = p2 = p3 = rep(0,sim_size)

P1 = P2 = P3 = rep(0,sim_size)

r1 = r2 = r3 = rep(0, sim_size)

R1 = R2 = R3 = rep(0,sim_size)
```

A loop is created to:
Repeat the following 2000 times: 1. generate predicted y values based on true coefficients and error variance and update a data frame containing predicted y values and the fixed predictors.
1. fit an MLR model to this updated data.
1. extract and save statistics from the results.

Significant Model Simulations
$\sigma_1 = 1$

```r
for(i in 1:sim_size){
  #1. predict n y values
  new_data = update.y(x1, x2, x3, B0, B1, B2, B3, sigma = 1)
  #fit a model to the simulated results
  mdl = lm(y ~ ., data = new_data)
  #collect summary statistics
  mdl_sum = summary(mdl)
  f = mdl_sum$fstatistic
  f1[i] = f[1]
  P1[i] = pf(f[1], f[2], f[3], lower.tail = FALSE)
  R1[i] = mdl_sum$r.squared
}
```

$\sigma_2 = 5$

```r
for(i in 1:sim_size){
  #1. predict n y values
  new_data = update.y(x1, x2, x3, B0, B1, B2, B3, sigma = 5)
  #fit a model to the simulated results
  mdl = lm(y ~ ., data = new_data)
  #collect summary statistics
  mdl_sum = summary(mdl)
  f = mdl_sum$fstatistic
  F2[i] = f[1]
  P2[i] = pf(f[1], f[2], f[3], lower.tail = FALSE)
  R2[i] = mdl_sum$r.squared
}
```

$\sigma_2 = 10$

```r
for(i in 1:sim_size){
  #1. predict n y values
  new_data = update.y(x1, x2, x3, B0, B1, B2, B3, sigma = 10)
  #fit a model to the simulated results
  mdl = lm(y ~ ., data = new_data)
  #collect summary statistics
  mdl_sum = summary(mdl)
  f = mdl_sum$fstatistic
  F3[i] = f[1]
  P3[i] = pf(f[1], f[2], f[3], lower.tail = FALSE)
  R3[i] = mdl_sum$r.squared
}
```

Non-significant Model Simulation

Again, this means that for the next three simulations the only difference from the above three is the coefficients of the true model.

$\sigma_1 = 1$

```r
#
for(i in 1:sim_size){
  #1. predict n y values
  new_data = update.y(x1, x2, x3, b0, b1, b2, b3, sigma = 1)
  #fit a model to the simulated results
  mdl = lm(y ~ ., data = new_data)
  #collect summary statistics
  mdl_sum = summary(mdl)
  f = mdl_sum$fstatistic
  f1[i] = f[1]
  p1[i] = pf(f[1], f[2], f[3], lower.tail = FALSE)
  r1[i] = mdl_sum$r.squared
}
```

$\sigma_2 = 5$

```r
for(i in 1:sim_size){
  #1. predict n y values
  new_data = update.y(x1, x2, x3, b0, b1, b2, b3, sigma = 5)
  #fit a model to the simulated results
  mdl = lm(y ~ ., data = new_data)
  #collect summary statistics
  mdl_sum = summary(mdl)
  f = mdl_sum$fstatistic
  f2[i] = f[1]
  p2[i] = pf(f[1], f[2], f[3], lower.tail = FALSE)
  r2[i] = mdl_sum$r.squared
}
```

$\sigma_2 = 10$

```r
for(i in 1:sim_size){
  #1. predict n y values
  new_data = update.y(x1, x2, x3, b0, b1, b2, b3, sigma = 10)
  #fit a model to the simulated results
  mdl = lm(y ~ ., data = new_data)
  #collect summary statistics
  mdl_sum = summary(mdl)
  f = mdl_sum$fstatistic
  f3[i] = f[1]
  p3[i] = pf(f[1], f[2], f[3], lower.tail = FALSE)
  r3[i] = mdl_sum$r.squared
}
```

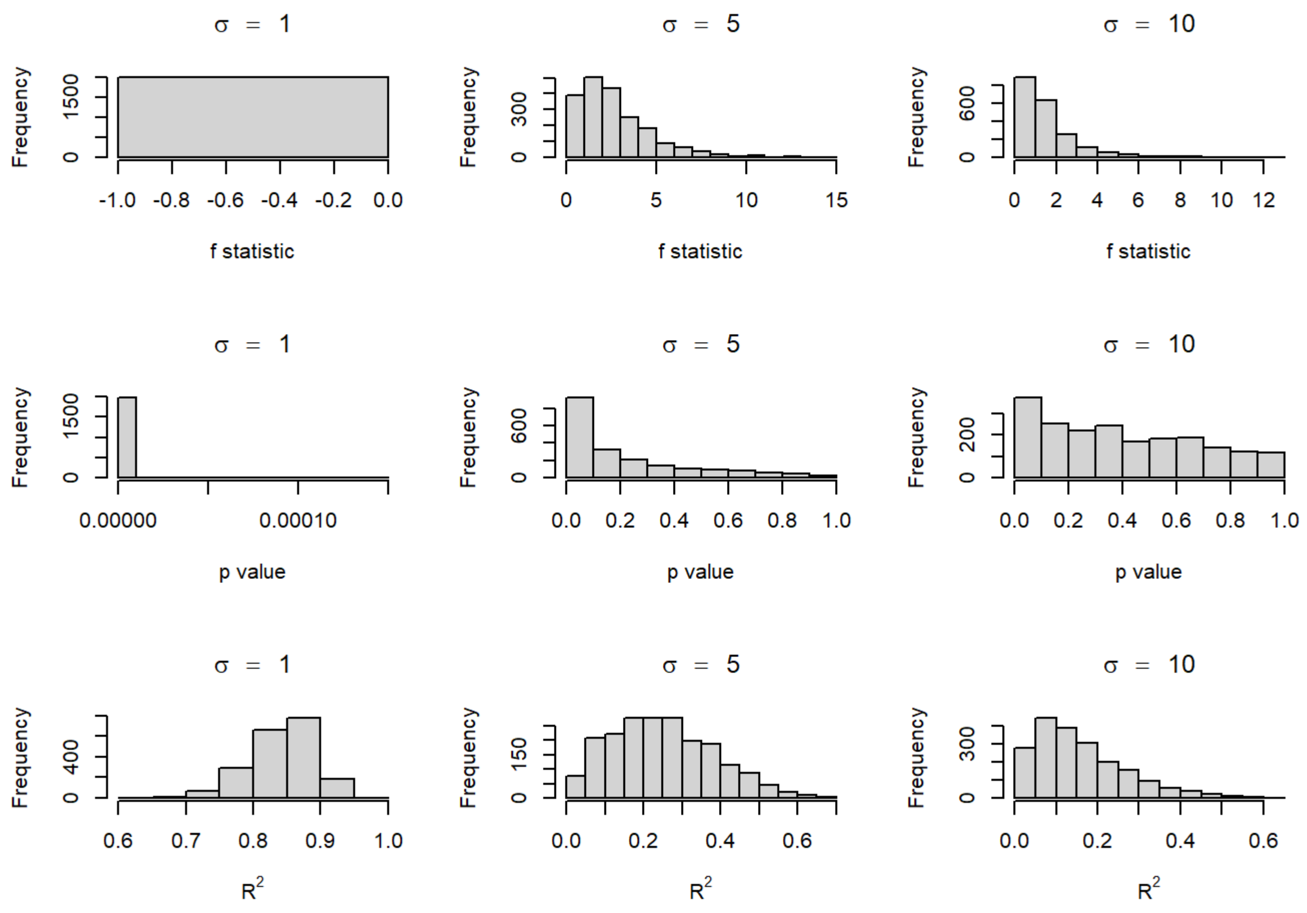
Results

The plots are organized in a a 3x3 grid the significant model and another for the non-significant model for easy comparison.

The first, second, and third rows display F statistic, p-value, and R^2 plots respectively as sigma increases from plots on the left to those on the right.

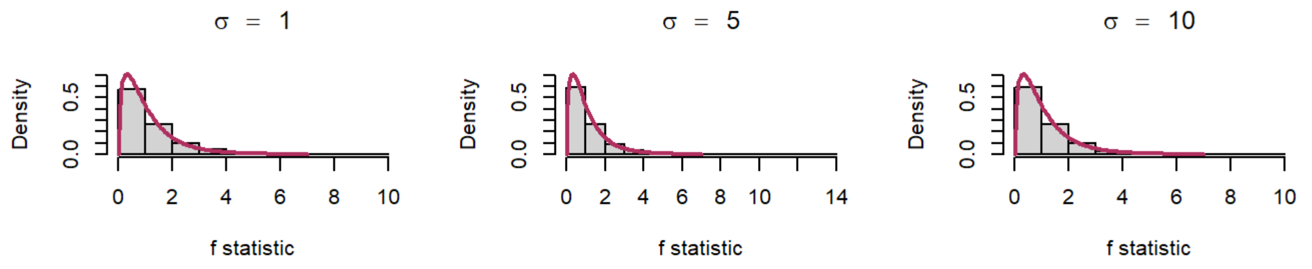
Significant Model:

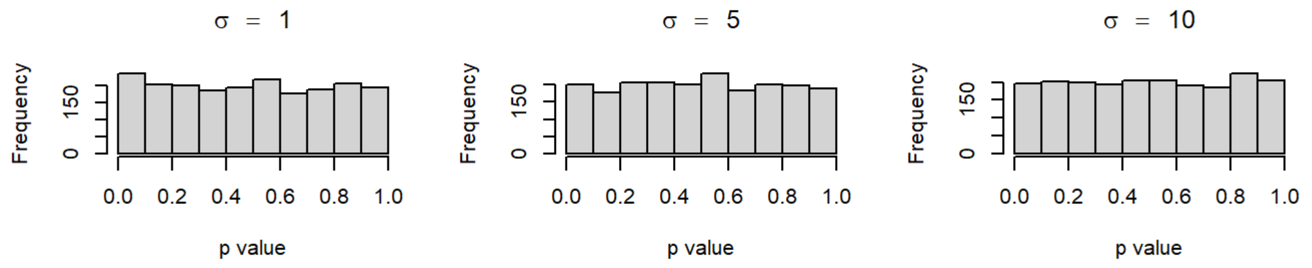
```r
par(mfrow=c(3,3))

hist(F1,
     xlab = "f statistic",
     main = TeX(r'($\sigma \ = \ 1$)'))
hist(F2,
     xlab = "f statistic",
     main = TeX(r'($\sigma \ = \ 5$)'))
hist(F3,
     xlab = "f statistic",
     main = TeX(r'($\sigma \ = \ 10$)'))

hist(P1,
     xlab = "p value",
     main = TeX(r'($\sigma \ = \ 1$)'))
hist(P2,
     xlab = "p value",
     main = TeX(r'($\sigma \ = \ 5$)'))
hist(P3,
     xlab = "p value",
     main = TeX(r'($\sigma \ = \ 10$)'))

hist(R1,
     xlab = TeX(r'($R^2$)'),
     main = TeX(r'($\sigma \ = \ 1$)'))
hist(R2,
     xlab = TeX(r'($R^2$)'),
     main = TeX(r'($\sigma \ = \ 5$)'))
hist(R3,
     xlab = TeX(r'($R^2$)'),
     main = TeX(r'($\sigma \ = \ 10$)'))
```



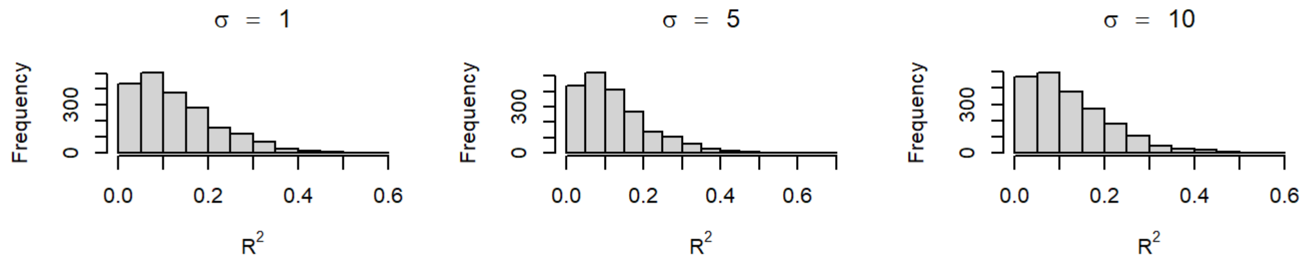
Non-significant Model:

```r
par(mfrow=c(3,3))

p = 4
m = seq(0, 7, 0.05)
fun = df(m, p-1, n-p)

hist(f1,
     freq = FALSE,
     ylim = range(fun),
     xlab = "f statistic",
     main = TeX(r'($\sigma \ = \ 1$)'))
lines(m, fun, col = "maroon", lwd = 2)

hist(f2,
     freq = FALSE,
     ylim = range(fun),
     xlab = "f statistic",
     main = TeX(r'($\sigma \ = \ 5$)'))
lines(m, fun, col = "maroon", lwd = 2)

hist(f3,
     freq = FALSE,
     ylim = range(fun),
     xlab = "f statistic",
     main = TeX(r'($\sigma \ = \ 10$)'))
lines(m, fun, col = "maroon", lwd = 2)

hist(p1,
     xlab = "p value",
     main = TeX(r'($\sigma \ = \ 1$)'))
hist(p2,
     xlab = "p value",
     main = TeX(r'($\sigma \ = \ 5$)'))
hist(p3,
     xlab = "p value",
     main = TeX(r'($\sigma \ = \ 10$)'))

hist(r1,
     xlab = TeX(r'($R^2$)'),
     main = TeX(r'($\sigma \ = \ 1$)'))
hist(r2,
     xlab = TeX(r'($R^2$)'),
     main = TeX(r'($\sigma \ = \ 5$)'))
hist(r3,
     xlab = TeX(r'($R^2$)'),
     main = TeX(r'($\sigma \ = \ 10$)'))
```





#



Discussion

Significant Model:

The F statistc's dependence on σ (noise).

As sigma increases the F statistic distribution narrows. This is because the denominator of the F statistic, the MSE, is an unbiased estimator of error variance. As the true variance increases, our simulated MSE increased, so the F statistic tends to be smaller.

The p-value's dependence on σ (noise).

As sigma increases the p-value distribution decreases. A smaller p-value for testing the significance of the regression means a test that is more likely to reject a null hypothesis that the regression is not significant. It is intuitive that more noise, a larger σ, would make noticing the signal, that the model is significant, less likely and this corresponds with a larger p-values.

The R^2's dependence on σ (noise).

As sigma increases the R^2 distribution moves to the left. This makes intuitive sense because we would expect that as the noise in the model increases the proportion of noise explained by the model, R^2, would decrease if everything is held equal, as it was.

Simulation Study 2: Using RMSE for Selection?

Introduction

In homework we saw how Test RMSE can be used to select the "best" model. In this simulation study we will investigate how well this procedure works. Since splitting the data is random, we do not expect it to work correctly each time. We could get unlucky. But averaged over many attempts, we should expect it to select the appropriate model.

We will simulate from the model

$$Y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \beta_4 x_{i4} + \beta_5 x_{i5} + \beta_6 x_{i6} + \epsilon_i$$

where $\epsilon_i \sim N(0, \sigma^2)$ and

- $\beta_0 = 0$,
- $\beta_1 = 3$,
- $\beta_2 = -4$,
- $\beta_3 = 1.6$,
- $\beta_4 = -1.1$,
- $\beta_5 = 0.7$,
- $\beta_6 = 0.5$.

We will consider a sample size of 500 and three possible levels of noise. That is, three values of σ.

- $n = 500$
- $\sigma \in (1, 2, 4)$

Use the data found in `study_2.csv` (study_2.csv) for the values of the predictors. These should be kept constant for the entirety of this study. The `y` values in this data are a blank placeholder.

Each time you simulate the data, randomly split the data into train and test sets of equal sizes (250 observations for training, 250 observations for testing).

For each, fit **nine** models, with forms:

- `y ~ x1`
- `y ~ x1 + x2`
- `y ~ x1 + x2 + x3`
- `y ~ x1 + x2 + x3 + x4`
- `y ~ x1 + x2 + x3 + x4 + x5`
- `y ~ x1 + x2 + x3 + x4 + x5 + x6`, the correct form of the model as noted above
- `y ~ x1 + x2 + x3 + x4 + x5 + x6 + x7`
- `y ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8`
- `y ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9`

For each model, calculate Train and Test RMSE.

$$\text{RMSE}(\text{model}, \text{data}) = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}$$

Repeat this process with 1000 simulations for each of the 3 values of σ. For each value of σ, create a plot that shows how average Train RMSE and average Test RMSE changes as a function of model size. Also show the number of times the model of each size was chosen for each value of σ.

Done correctly, you will have simulated the y vector $3 \times 1000 = 3000$ times. You will have fit $9 \times 3 \times 1000 = 27000$ models. A minimal result would use 3 plots. Additional plots may also be useful.

Methods:

Simulation size is set.

```
sim_size = 1000
```

The data is loaded.

```
data = read.csv("study_2.csv")
```

The sample size is set.

```
n = 500
```

Vectors are created to store the true beta values and the true sigma values.

```
beta = as.vector(c(0, 3, -4, 1.6, -1.1, 0.7, 0.5, 0, 0, 0))
sig_vtr = c(1, 2, 4)
```

The seed is set for the study.

```
birthday = 19781020
set.seed(birthday)
```

A function is created that takes the predictors, true coefficient values β, and true σ and returns a data frame including the existing predictors and the predictions generated from the true model.

```
sim_mlr = function(x, beta, sigma) {
  epsilon = rnorm(n, mean = 0, sd = sigma)
  y = X %*% beta + epsilon
  data.frame(y, x)
}
```

Create RMSE function that that takes simulated values and data set containing the actual response values.

```
rmse = function(fitted.values, data){
  resids = fitted.values - data$y
  squared.error = resids^2
  mse = mean(squared.error)
  sqrt(mse)
}
```

An X matrix is created so the matrix representation can be used to calculate predicted values.

```
X = as.matrix(cbind(rep(1, 500), data[, -1]))
```

Three matrices are created to store results. 1. Will store the number of times models of each size are selected for each sigma by allowing row the represent the number of predictors and columns to represent $\sigma = 1,\ \sigma = 2,\ and\ \sigma = 4$, respectfully.

```
mdl_seln = matrix(rep(0, 27), nrow = 9, ncol = 3)
mean_tst_rmse  = matrix(rep(0, 27), nrow = 9, ncol = 3)
mean_trn_rmse  = matrix(rep(0, 27), nrow = 9, ncol = 3)
```

A nested loop is used. The outer loop will cycle through values of σ and the inner loop with cycle through of the 1000 simulations.

Within each iteration of the outer loop 1000 x 9 matrices are created to hold test and train RMSE until 1000 inner loop iterations are completed. Then these are averaged over the 1000 rows to calculate average RMSE for each model size from from one predictor to nine and store them in `mdl_seln` , in 9 x 3 matrix defined above. It contains one column for each value of σ and one row for each model size from one predictor to nine.

```r
for(i in 1:3){ #loop through sigma = 1, 2, 4
  sigma = sig_vtr[[i]]

  rmse_trn = matrix(rep(0, 9000), nrow = 1000, ncol = 9)
  rmse_tst = matrix(rep(0, 9000), nrow = 1000, ncol = 9)
  for(j in 1:sim_size){  #repeat simulations

    # simulate response from TRUE model and save the new predicted responses        and the cons
tant values of each predictor.
    sim_resp = sim_mlr(X, beta, sigma)[,-2]

    # SPLIT these results into training and testing sets
    data_trn_idx = sample(1:nrow(data), 250)
    sim_trn = sim_resp[data_trn_idx, ]  #training data
    sim_tst = sim_resp[-data_trn_idx, ] #test data

    ##FIT##

    # In each of the 9 iteration of the next loop
      # A model with k predictors is fit to the training data
      # Then only using this model to predict
      # test RMSE is calculated from predictions based on test predictors and       # simulated
 actual values from the test data and
      # training RMSE is calculated from predictions based on training              # predictors
and          simulated actual values from the test data
    # Each time save test RMSE = y_test(train model) - X_test and train RMSE = y_train() - test_
train
    for(k in 1:9){
      #FIT model k to sim_trn with k predictors
      lm_trn = lm(y ~ ., data = data.frame(sim_trn[, 1:(1+k)]))
      #calculate training data RMSE. Calculate RMSE from predictions from predictors
      #in training data and actual response values from training data
      rmse_trn[j, k] = rmse(
        fitted.values = predict(lm_trn),
        data = sim_trn[, 1:(1+k)])
      #calculate test data RMSE. Calculate RMSE from predictions from predictors
      #in test data and actual response values from test data.
      rmse_tst[j, k] = rmse(
        fitted.values = predict(lm_trn, newdata = data.frame(sim_tst[, 1:(1+k)])),
        data = sim_tst[, 1:(1+k)]
      )
    }
    # the model with the lowest RMSE is selected so the count in the cell corresponding to the r
ow number          # representing the number of predictors of that model and column 1, 2, or 3 c
orresponding to sigma of 1, 2,     # or 4 is incremented by one

    mdl_seln[which.min(rmse_tst[j,]), i] = mdl_seln[which.min(rmse_tst[j,]), i] + 1
  } # end simulations loop
  #save mean RMSE for each model size
  mean_trn_rmse[,i] = colMeans(rmse_trn, na.rm = TRUE)
  mean_tst_rmse[, i] = colMeans(rmse_tst, na.rm = TRUE)
} # end of loop for sigma value
```

Results:

The number of times the model with number of predictors = row number and column = sigma of 1, 2, or 4 was chosen is displayed in the following table.

Number of times the model of each size was chosen at each sigma

	Sigma…1	Sigma…2	Sigma…3
1 Predictors	0	0	0
2 Predictors	0	0	1
3 Predictors	0	2	13
4 Predictors	0	1	12
5 Predictors	0	2	8
6 Predictors	56	56	32
7 Predictors	21	16	14
8 Predictors	8	14	13
9 Predictors	15	9	7

```
# put the next two plots side-by-side
par(mfrow = c(1,2))

kable(data.frame(mean_trn_rmse), caption = 'Training')
```

Training

	Sigma…1	Sigma…2	Sigma…3
1 Predictors	0.2564	0.3095	0.4637
2 Predictors	0.1587	0.2351	0.4167
3 Predictors	0.1163	0.2086	0.4007
4 Predictors	0.1126	0.2062	0.3991
5 Predictors	0.1111	0.2052	0.3979
6 Predictors	0.0985	0.1981	0.3937
7 Predictors	0.0983	0.1977	0.3928
8 Predictors	0.0981	0.1972	0.3920

	Sigma…1	Sigma…2	Sigma…3
9 Predictors	0.0978	0.1967	0.3908

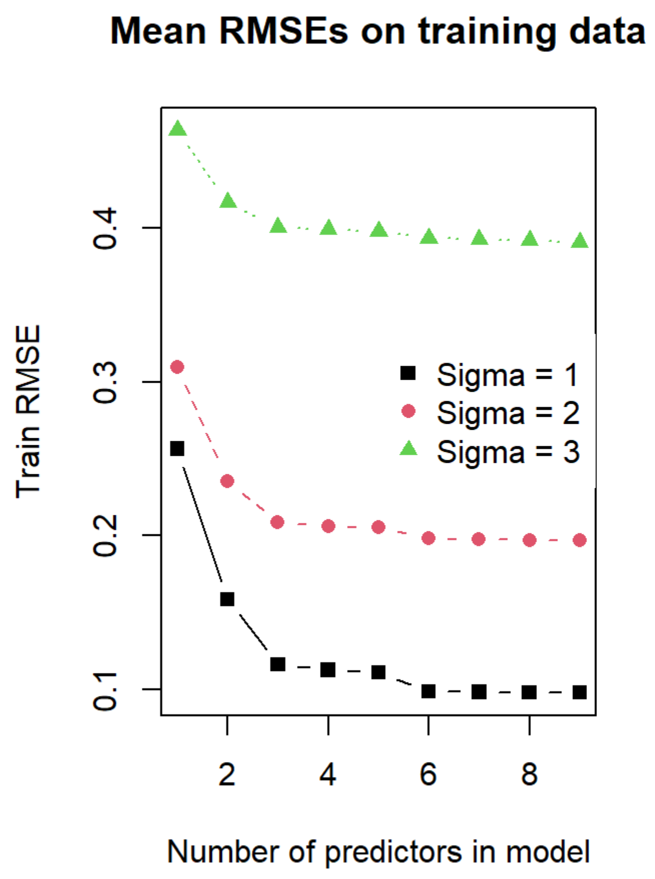
```
kable(data.frame(mean_tst_rmse), caption = 'Testing')
```

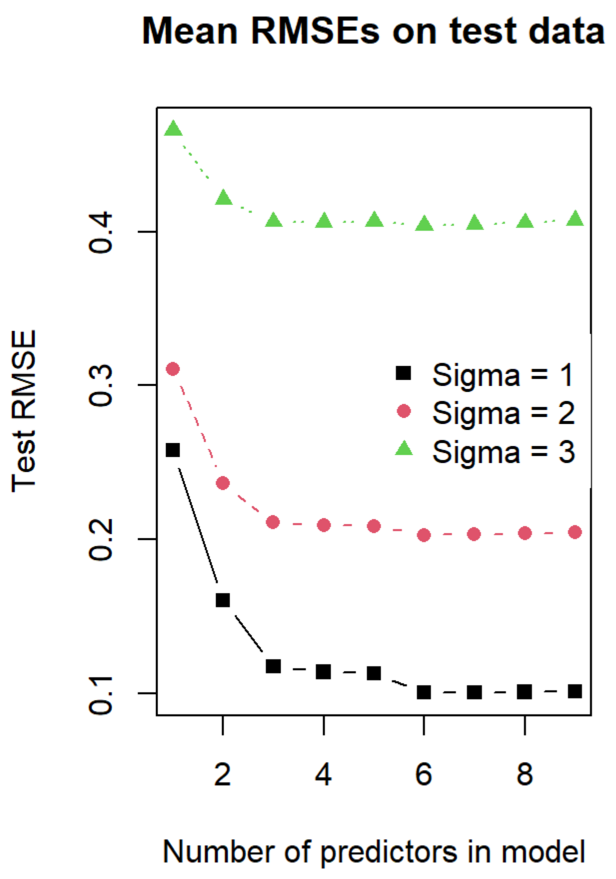
Testing

	Sigma…1	Sigma…2	Sigma…3
1 Predictors	0.2581	0.3108	0.4657
2 Predictors	0.1606	0.2363	0.4207
3 Predictors	0.1177	0.2110	0.4063
4 Predictors	0.1142	0.2093	0.4059
5 Predictors	0.1130	0.2088	0.4066
6 Predictors	0.1007	0.2030	0.4040
7 Predictors	0.1009	0.2033	0.4048
8 Predictors	0.1012	0.2039	0.4057
9 Predictors	0.1014	0.2045	0.4073

For each value of sigma, average Train RMSE and Test RMSE are plotted against model size (9 models).

```r
# put the rest of the plots side-by-side
par(mfrow = c(1,2))

#plot mean training RMSEs
matplot(mean_trn_rmse,
     type = c("b"),
     pty = c(1:3),
     pch = c(15:17),
     col = c(1:3),
     main = 'Mean RMSEs on training data',
     xlab = "Number of predictors in model",
     ylab= "Train RMSE")
legend("right", legend = colnames(mean_trn_rmse),
       pch = c(15:17),
       col = c(1:3),
       box.lty = 0)

#plot mean test RMSEs
matplot(mean_tst_rmse,
     type = c("b"),
     pty = c(1:3),
     pch = c(15:17),
     col = c(1:3),
     main = 'Mean RMSEs on test data',
     xlab = "Number of predictors in model",
     ylab= "Test RMSE")
legend("right", legend = colnames(mean_tst_rmse),
       pch = c(15:17),
       col = c(1:3),
       box.lty = 0)
```





Discussion:

- As we can see from the number of times each model was selected at different values of sigma, ie levels of noise. The method of choosing the the model with the lowest test RMSE does not **always** select the true model, but it does so more often than not.
- The greater the σ or noise the more difficult it is to find the signal or true model using minimum RMSE.
- RMSE reduces as the number of parameters increase whether predicting using test or training data.
- As sigma or true error standard deviation increases the RMSE increases because both measure noise or variance.

Simulation Study 3: Power

Introduction

In this simulation study we will investigate the **power** of the significance of regression test for simple linear regression.

$H_0 : \beta_1 = 0$ vs $H_1 : \beta_1 \neq 0$

Recall, we had defined the *significance* level, α, to be the probability of a Type I error.

$$\alpha = P[\text{Reject } H_0 \mid H_0 \text{ True}] = P[\text{Type I Error}]$$

Similarly, the probability of a Type II error is often denoted using β; however, this should not be confused with a regression parameter.

$$\beta = P[\text{Fail to Reject } H_0 \mid H_1 \text{ True}] = P[\text{Type II Error}]$$

Power is the probability of rejecting the null hypothesis when the null is not true, that is, the alternative is true and β_1 is non-zero.

$$\text{Power} = 1 - \beta = P[\text{Reject } H_0 \mid H_1 \text{ True}]$$

Essentially, power is the probability that a signal of a particular strength will be detected. Many things affect the power of a test. In this case, some of those are:

- Sample Size, n
- Signal Strength, β_1
- Noise Level, σ
- Significance Level, α

We'll investigate the first three.

To do so we will simulate from the model

$$Y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

where $\epsilon_i \sim N(0, \sigma^2)$.

For simplicity, we will let $\beta_0 = 0$, thus β_1 is essentially controlling the amount of "signal." We will then consider different signals, noises, and sample sizes:

- $\beta_1 \in (-2, -1.9, -1.8, \ldots, -0.1, 0, 0.1, 0.2, 0.3, \ldots 1.9, 2)$
- $\sigma \in (1, 2, 4)$
- $n \in (10, 20, 30)$

We will hold the significance level constant at $\alpha = 0.05$.

Use the following code to generate the predictor values, `x`: values for different sample sizes.

```
x_values = seq(0, 5, length = n)
```

For each possible β_1 and σ combination, simulate from the true model at least 1000 times. Each time, perform the significance of the regression test. To estimate the power with these simulations, and some α, use

$$\hat{\text{Power}} = \hat{P}[\text{Reject } H_0 \mid H_1 \text{ True}] = \frac{\text{\# Tests Rejected}}{\text{\# Simulations}}$$

It is *possible* to derive an expression for power mathematically, but often this is difficult, so instead, we rely on simulation.

Create three plots, one for each value of σ. Within each of these plots, add a "power curve" for each value of n that shows how power is affected by signal strength, β_1.

Potential discussions:

- How do n, β_1, and σ affect power? Consider additional plots to demonstrate these effects.
- Are 1000 simulations sufficient?

An additional tip:

- Search online for examples of power curves to give you inspiration for how you might construct your own plots here. You'll find both two-sided and one-sided power curves. Based on the way you're asked to construct the β_1 vector, you should be able to figure out which type is appropriate here.

Methods

- Each of three levels of noise, true error standard deviation, $\sigma \in (1,\ 2,\ 4)$ is considered.
- For each of these, three sample sizes $n \in (10,\ 20,\ 40)$ is considered and finally for each of these 41 levels of signal or regression coefficient - $\beta_1 \in (-2, -1.9, -1.8, \ldots, -0.1, 0, 0.1, 0.2, 0.3, \ldots 1.9, 2)$. s

The parameters used are defined as vectors where possible.

```
S = c(1, 2, 4)
N = c(10, 20, 30)
B = seq(-2, 2, 0.1)
alpha = 0.05
num_sims = 1000
```

A plot for each σ is desired of Power in response to signal, β_1 so we will create three zero matrices of 41 rows corresponding to β values by 3 columns corresponding to σ values to store the results.

After each simulation a significance of regression test is performed and its p-value is recorded until the completion of 1000 simulations. Then these are used to calculate power using the equation presented in the introduction. Each of the $41\ x\ 3\ =\ 123$ cells will contain contain these power values.

```
sig1 = sig2 = sig4 = matrix(rep(0, 41 * 3),
                            nrow = 41,
                            ncol = 3
                            )
sig = list(sig1, sig2, sig4)
colnames(sig[[1]]) = colnames(sig[[2]]) = colnames(sig[[3]]) =
  c("n10", "n20", "n30")
a = rep("beta = ", 41)
b = as.character(seq(-2, 2, by = 0.1))
c = paste(a, b)
rownames(sig[[1]]) =
rownames(sig[[2]]) =
rownames(sig[[3]]) = c
```

A seed is set and used throughout the project.

```
birthday = 19781020
set.seed(birthday)
```

A function is created that takes the predictor, true coefficient value, and true sigma and returns a data frame with the predictor and predictions generated from the true model.

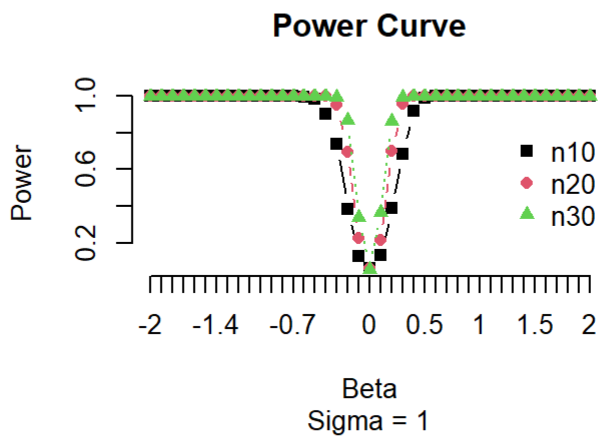
```r
sim_slr = function(x, beta_1, sigma) {
  n = length(x)
  epsilon = rnorm(n, mean = 0, sd = sigma)
  y = beta_1 * x + epsilon
  data.frame(y, x)
}
```

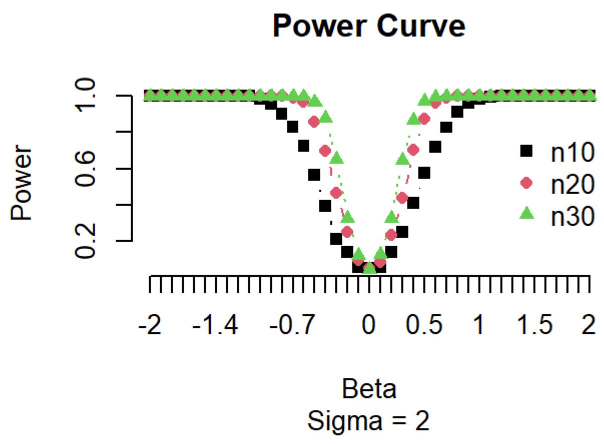
```r
p = vector() #pvalue vector

for(s in S){ #loop through sigma
  for(n in N){ #loop through sample sizes
    for(b in B){
      for(i in 1:num_sims){  #repeat simulations
        #generate predictor values as instructed
        x_values = x = seq(0, 5, length = n)
        #simulate prediction results
        sim_data = sim_slr(x, b, s)
        #fit the true model, with beta0 = 0, ie no intercept
        mdl = lm(y ~ 0 + x, data = sim_data)
        p[i] = summary(mdl)$coef[1, 4]
      }#end i
      sig[[which(S == s)]][which(B == b), which(N == n)] = mean(p < alpha)
      sprintf(p , fmt = '%#.3f')
      sprintf(p < alpha , fmt = '%#.3f')
      sprintf(mean(p < alpha) , fmt = '%#.3f')
    }#end b
  }#end n
}#end s
```

Results

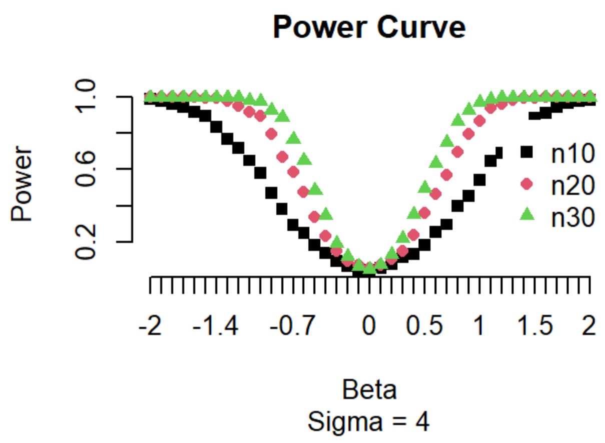
###Power Curves, plots of power on signal strength, ie β with varying standard error σ and sample size, n.

```
par(mfrow = c(2,2))

matplot(sig[[1]],
     type = c("b"),
     pty = c(1:3),
     pch = c(15:17),
     col = c(1:3),
     main = 'Power Curve',
     sub = 'Sigma = 1',
     xlab = "Beta",
     ylab= "Power",
     axes = FALSE)
legend("right", legend = colnames(sig[[1]]),
       pch = c(15:17),
       col = c(1:3),
       box.lty = 0)
axis(side=1,at=1:length(B),labels=B)
axis(2)

matplot(sig[[2]],
     type = c("b"),
     pty = c(1:3),
     pch = c(15:17),
     col = c(1:3),
     main = 'Power Curve',
     sub = 'Sigma = 2',
     xlab = "Beta",
     ylab= "Power",
     axes = FALSE)
legend("right", legend = colnames(sig[[2]]),
       pch = c(15:17),
       col = c(1:3),
       box.lty = 0)
axis(side=1,at=1:length(B),labels=B)
axis(2)

matplot(sig[[3]],
     type = c("b"),
     pty = c(1:3),
     pch = c(15:17),
     col = c(1:3),
     main = 'Power Curve',
     sub = 'Sigma = 4',
     xlab = "Beta",
     ylab= "Power",
     axes = FALSE)
legend("right", legend = colnames(sig[[3]]),
       pch = c(15:17),
       col = c(1:3),
       box.lty = 0)
axis(side=1,at=1:length(B),labels=B)
axis(2)
```





##



Discussion

- How do n, β_1, and σ affect power?

n

- As sample size increases power increases. The impact of sample size is greatest when β is not near 0 nor near $-2\ or\ 2$.

β

- As the $|\beta|$ increases the signal increases so the power or ability to find a signal increases but reaches an asymptote at larger values.

σ

As noise or σ increases the ability to detect the signal or power decreases.

Number of Simulations:

- Whether 1000 simulations is sufficient depends on how accurate your power curve needs to be, but it will start making less and less of a different at some point and the increased accuracy may not be worth the time it takes to perform the simulations.