Class 1 Homework - Linear Equations

Overview

Create your own RMD *(Again RMD tutorials are here: https://rmarkdown.rstudio.com/.)* Turn in both your RMD and PDF files.

Least Squares

Understanding how parameter values are estimated is foundational to modeling, and least squares *(LS)* is where it began. LS provides a unique, optimal solution but becomes intractable and unstable in complex problems.

Least Sqaures is a derivative based optimization. Differential calculus is common in analytics, and applications include:

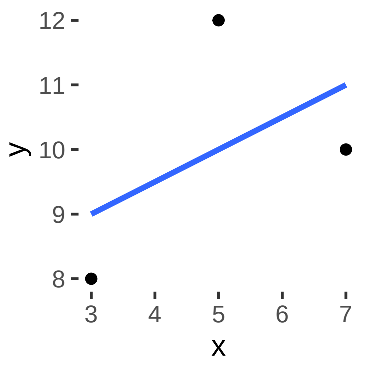
- **Function Optimization.** Here, we use a derivative to optimize the cost *(error)* function, also with gradient descent *(a more computational approach to cost function optimization)*. And we'll use derivatives to optimize objective functions in non-linear regression later.
- **Elimination.** Elimination is a basic approach to solving systems of equations and common in a wide range of advanced modeling algorithms *(we'll see this in QR decomposition)*.

Use the following Data:

```r
LSData = data.frame(x = c(3, 5, 7), y = c(8, 12, 10))
knitr::kable(data.frame(LSData)) %>%
  kable_styling(full_width = F, bootstrap_options = "striped", font_size = 9)
```

x	y
3	8
5	12
7	10

```r
p = ggplot(LSData, aes(x = x, y = y)) +
  geom_point() +
  geom_smooth(method = "lm", se = F) +
  theme(panel.background = element_rect(fill = "white"))
p
```



Now, write the partial derivatives with regard to β_0 and β_1, e.g.:

$\partial\beta_0 = x\beta_0 + x\beta_1 + x = x$
$\partial\beta_1 = x\beta_0 + x\beta_1 + x = x$

Then show your parameter estimates and reconcile to lm estimates

Generating Data

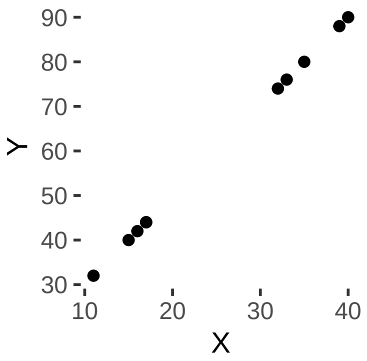
It's important that you learn to generate data. Generating data for models builds intuition by forcing you to comprehend data structure and define the "actual" parameters so you can tailor the data to your question and assess model performance.

Follow the approach below and generate some data *(note, we define the equation first - so we're defining the parameter and parameter values - i.e., we know what's "real")*. **Your equation must be unique - you can't copy and paste my code!**

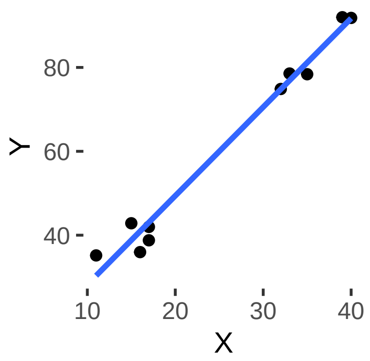
```
N = 10
Beta = 2
Alpha = 10

Data = data.frame(X = sample(seq(from = 10, to =40, by = 1), N, replace = TRUE))
Data = Data %>% mutate(Y =  Alpha + (X*Beta))

p = ggplot (aes(x = X, y = Y), data = Data) +
  geom_point() +
  theme(panel.background = element_rect(fill = "white"))
p
```



Since all the data was generated using the equation, we get a straight line *(duh!)*. But that's not veey real *(and it will actually cause models to fail because there's no variance / covariance matrix - which is what* *we use to calculate regression coefficients)*. So, lets add some random noise *(you can't use my distribution parameters - no copy and paste!)*:



Now, fit the model and show parameters *(look at your parameter estimate - why is it different from the values you set?, change the scale in the rnorm distribution and note the change in parameter values)*:

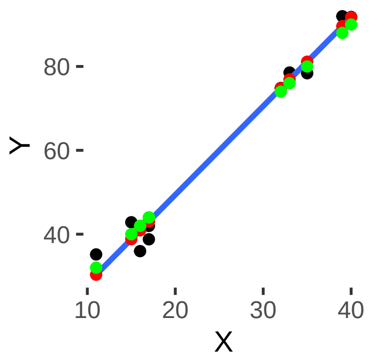
```
mFit <- lm(Y ~ X, data = Data)
knitr::kable(data.frame(round(mFit$coefficients,2))) %>%
  kable_styling(full_width = F, bootstrap_options = "striped", font_size = 9)
```

	round.mFit.coefficients..2.
(Intercept)	7.06
X	2.12

Use the lm predict function to create yhat values, and then use an equation to do the same *(do the green dots line up exactly on the regression line? Why?)*:

```
Data$yhat <- predict(mFit, Data)
Data$yhat2  = Alpha + (Data$X * Beta)

p = ggplot (aes(x = X, y = Y), data = Data) +
  geom_point() +
  geom_smooth(method = "lm", se = F) +
  geom_point(aes(x = X, y = yhat), color = "red") +
  geom_point(aes(x = X, y = yhat2), color = "green") +
  theme(panel.background = element_rect(fill = "white"))
p
```



Categorical Variables

Now generate a model with categorical variables *(use different categories and effects - no copy and paste)*:

```r
# now generate with categorical (discrete variables)

N2 = N*3
# Observations generated have to be multiple of the number of categories
# (R vectorizes the rnorm function and will insert the rows into the dataframe
# as long as the total is a multiple of the Model column values)

Data = data.frame(
  Category = c("Category1", "Category2", "Category3"),
  CEffect = c(0, 10, 20),
  X = sample(seq(from = 10, to =40, by = 1), N2, replace = TRUE))

Data = Data %>% mutate(Intercept =  (Alpha + CEffect)) %>%
                     mutate(Y = Intercept + (X * Beta)) %>%
                     mutate(Y = round(Y + rnorm(nrow(Data), 0, 5),0))

mFit2 <- lm(Y ~ Category + X, data = Data)

knitr::kable(data.frame(round(mFit2$coefficients,2))) %>%
  kable_styling(full_width = F, bootstrap_options = "striped", font_size = 9)
```

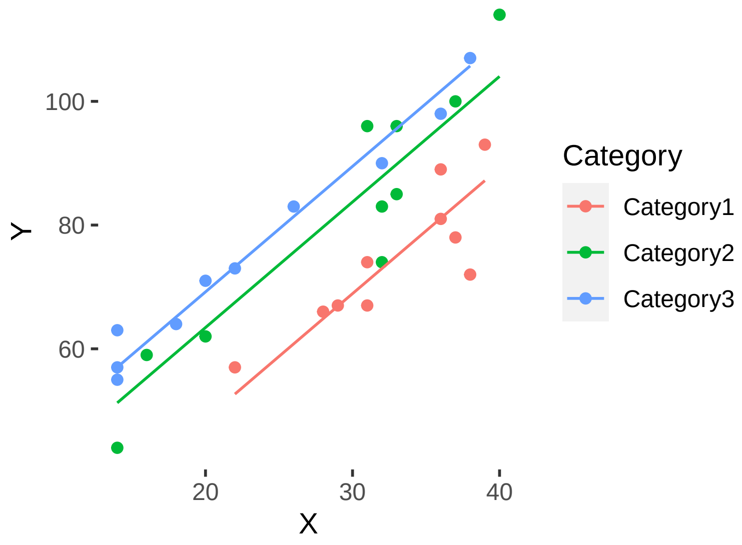
	round.mFit2.coefficients..2.
(Intercept)	8.04
CategoryCategory2	14.81
CategoryCategory3	20.57
X	2.03

Now plot the data with predicted values *(if you give ggplot a color parameter, it will apply that across all layers - e.g., geom_point and geom_line in this case)*:

```r
Data$yhat <- predict(mFit2, Data)

p = ggplot (aes(x = X, y = Y, color = Category), data = Data) +
  geom_point() +
  geom_line(aes(y = yhat)) +
  theme(panel.background = element_rect(fill = "white"))

p
```



Now solve using normal equations. Did you get the same thing as lm? Why? *(just write answers below)*

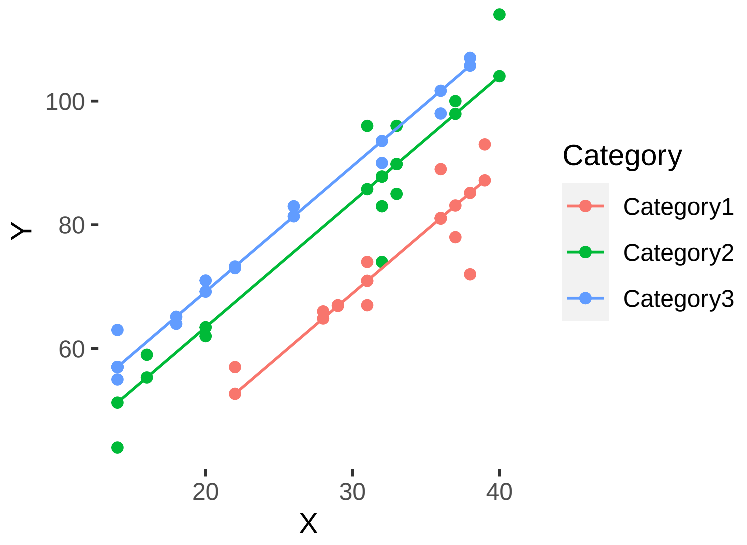
```
vY = as.matrix(select(Data, Y))
mX = model.matrix(Y ~ Category + X,  Data)
vBeta = solve(t(mX)%*%mX, t(mX)%*%vY) # solve using normal equations
knitr::kable(data.frame(round(vBeta,2))) %>%
  kable_styling(full_width = F, bootstrap_options = "striped", font_size = 9)
```

	Y
(Intercept)	8.04
CategoryCategory2	14.81
CategoryCategory3	20.57
X	2.03

Add your normal equation line to the plot

```
vBeta2 <- as.numeric(vBeta)
Data$neY <- t(vBeta2%*%t(mX))
# compare predictions using NE vs lm - should be the same

p = p +
  geom_point(aes(x = X, y = neY), data = Data) +
  theme(panel.background = element_rect(fill = "white"))
p
```



Category
Category1
Category2
Category3
Y
X
100
80
60
20
30
40