INFO 534 Homework 2

Rachel Sagers

2023-02-16

Q1

(a)

A cost function is used to measure how far the prediction made by the model is from the actual input data by calculating the difference between the two values. This measures how well the machine learning model is fit to our data. The full expression of the cost function is:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

(b)

The gradient descent algorithm finds the point where the cost function is minimized. The best estimates of the model parameters are at the cost function minimum.

(c)

```
#partial derivative function of theta0
dJ0 = function(theta0,theta1,m,x,y){
  yhat = theta0 + theta1*x
  dJ0 = (1/m)*sum(yhat-y)
  return(dJ0)
}

#partial derivative function of theta1
dJ1 = function(theta0,theta1,m,x,y){
  yhat = theta0 + theta1*x
  dJ1 = (1/m)*sum((yhat-y)*x)
  return(dJ1)
}

#cost function J
J = function(theta0,theta1,m,x,y){
  yhat = theta0 + theta1*x
  J_value = (1/(2*m))*sum((yhat-y)^2)
  return(J_value)
}
```

(d)

```
library(dplyr)
```

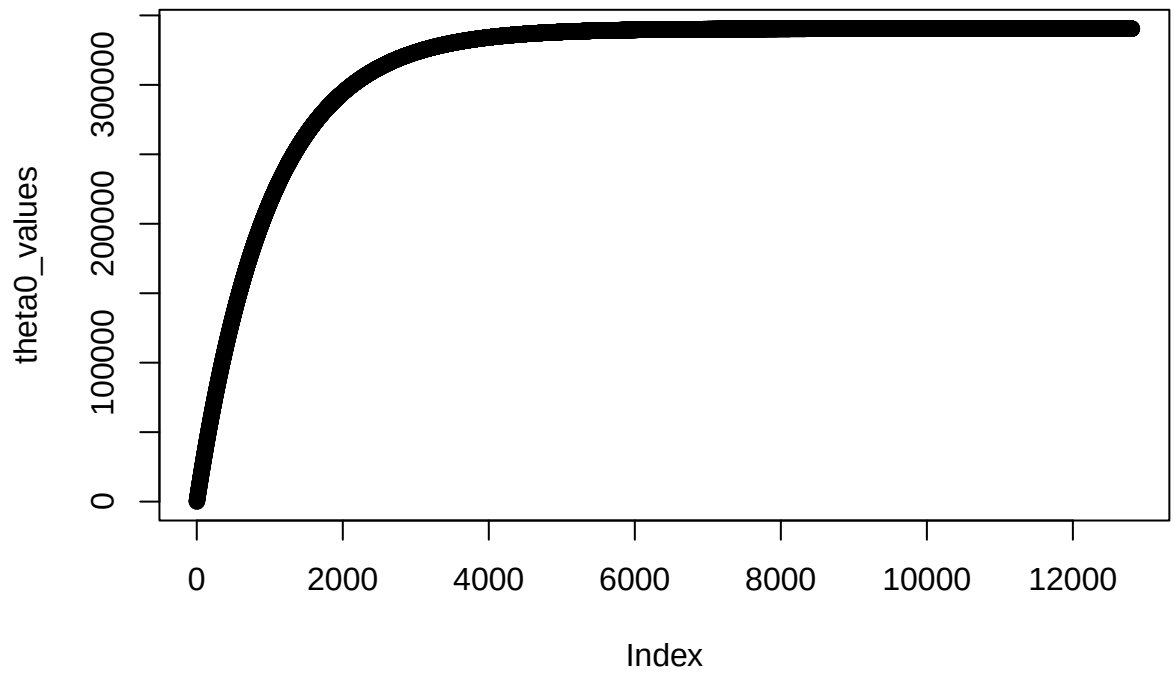
```
##
## Attaching package: 'dplyr'

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

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

```
#read in data
housing <- read.csv("housing.txt")

#feature scaling using dplyr mutate function
housing_scaled <- housing %>%
  mutate_at("size", scale)

x <- housing_scaled$size
y <- housing_scaled$price
m <- nrow(x)

#set convergence threshold and learning rate
convg_threshold = 10^-3
alpha = 0.001

initial_theta0 = 1 # initial guess of theta0
initial_theta1 = 2 # initial guess of theta1

initial_J = J(initial_theta0,initial_theta1,m,x,y) # initial value of the cost function

current_theta0 = initial_theta0
current_theta1 = initial_theta1
current_J = initial_J

theta0_values = c(current_theta0) # a vector to store updated theta0 values during iterations
theta1_values = c(current_theta1) # a vector to store updated theta1 values during iterations
J_values = c(current_J) # a vector to store cost function values during iterations

delta_J = 1 # an arbitrary value to start the algorithm

while (delta_J > convg_threshold) {
  new_theta0 = current_theta0 - alpha*dJ0(current_theta0,current_theta1,m,x,y)
  new_theta1 = current_theta1 - alpha*dJ1(current_theta0,current_theta1,m,x,y)
  new_J = J(new_theta0,new_theta1,m,x,y)

  theta0_values = c(theta0_values, new_theta0)
  theta1_values = c(theta1_values, new_theta1)
```

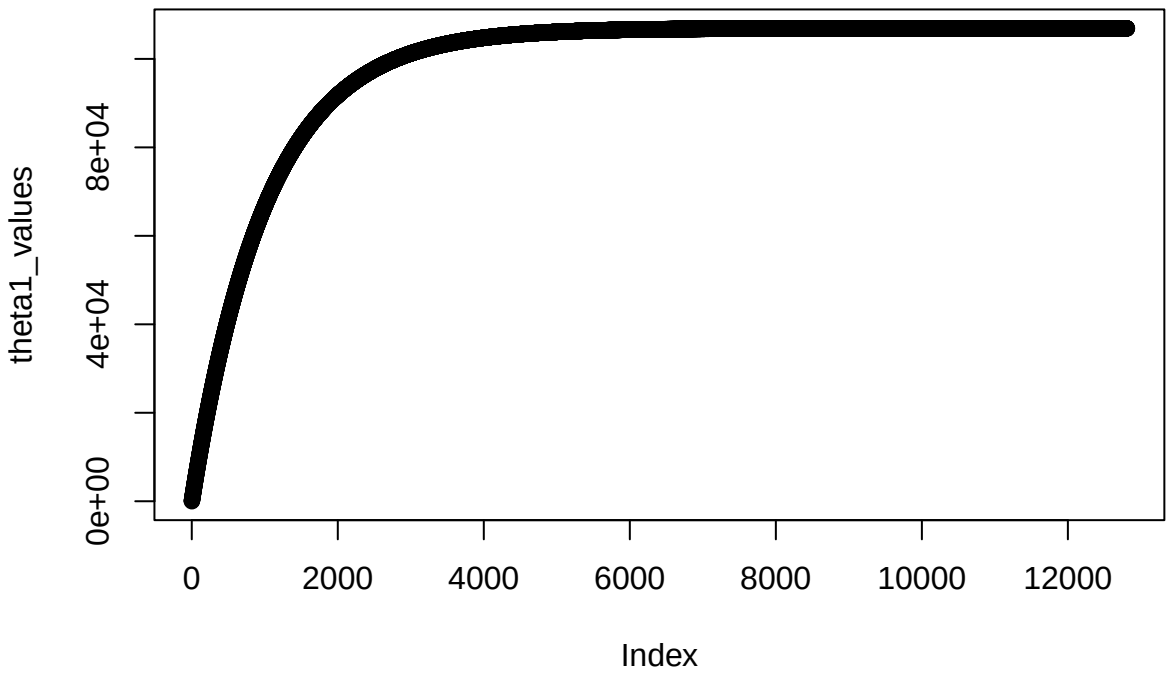
```
  J_values = c(J_values, new_J)

  delta_J = abs(new_J - current_J)

  ## update theta0, theta1, and J values ##
  current_J = new_J
  current_theta0 = new_theta0
  current_theta1 = new_theta1
}
```

(e)

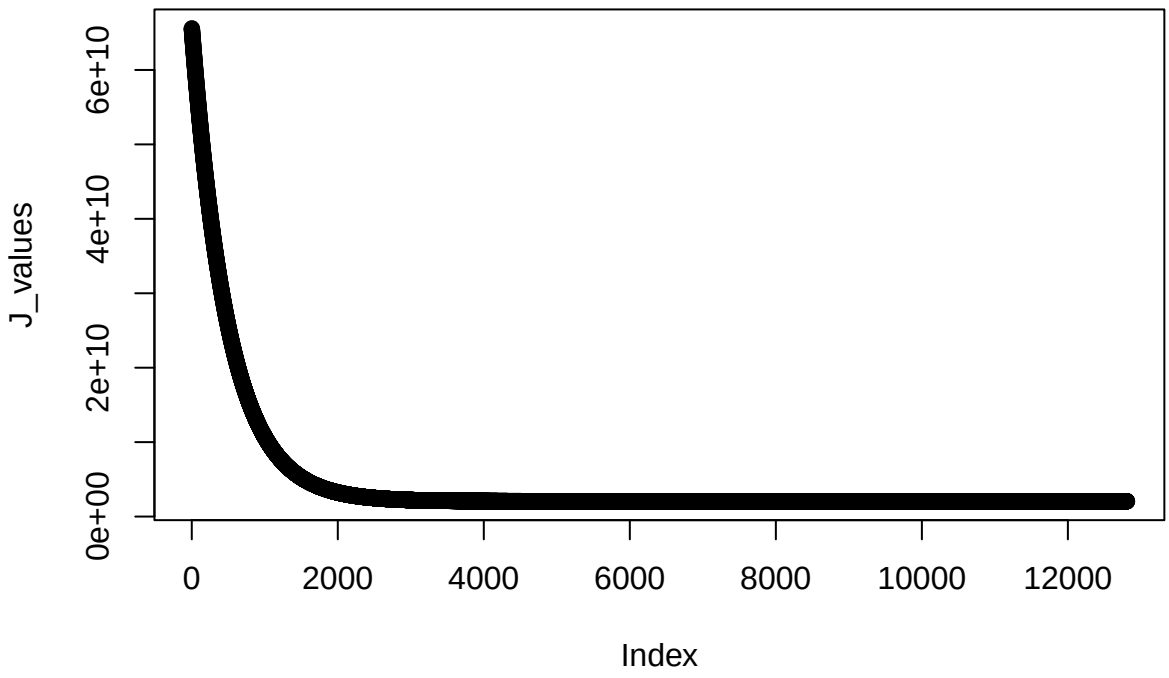
```
#plot of theta0 values at each iteration
plot(theta0_values)
```




```
#plot of theta1 values at each iteration
plot(theta1_values)
```




```
#plot of J values at each iteration
plot(J_values)
```



(f)

```
print(tail(theta0_values, n=1))
```

```
## [1] 340411.7
```

```
print(tail(theta1_values, n=1))
```

```
## [1] 106907.2
```

```
print(tail(J_values, n=1))
```

```
## [1] 2058132741
```

(g)

```
#read in data
housing_unscaled <- read.csv("housing.txt")

un_x <- housing_unscaled$size
```

```r
un_y <- housing_unscaled$price
un_m <- nrow(housing_unscaled)

#set convergence threshold and learning rate
un_convg_threshold <- 10^-3
un_alpha = 0.001

un_initial_theta0 = 1 # initial guess of theta0
un_initial_theta1 = 2 # initial guess of theta1

un_initial_J = J(un_initial_theta0,un_initial_theta1,un_m,un_x,un_y) # initial value of the cost functi

un_current_theta0 = un_initial_theta0
un_current_theta1 = un_initial_theta1
un_current_J = un_initial_J

un_theta0_values = c(un_current_theta0) # a vector to store updated theta0 values during iterations
un_theta1_values = c(un_current_theta1) # a vector to store updated theta1 values during iterations
un_J_values = c(un_current_J) # a vector to store cost function values during iterations

un_delta_J <- 1 # an arbitrary value to start the algorithm

while (un_delta_J > un_convg_threshold){
  un_new_theta0 = un_current_theta0 - un_alpha*dJ0(un_current_theta0,un_current_theta1,un_m,un_x,un_y)
  un_new_theta1 = un_current_theta1 - un_alpha*dJ1(un_current_theta0,un_current_theta1,un_m,un_x,un_y)
  un_new_J = J(un_new_theta0,un_new_theta1,un_m,un_x,un_y)

  un_theta0_values = c(un_theta0_values, un_new_theta0)
  un_theta1_values = c(un_theta1_values, un_new_theta1)
  un_J_values = c(un_J_values, un_new_J)

  un_delta_J = abs(un_new_J - un_current_J)

  ## update theta0, theta1, and J values ##
  un_current_J = un_new_J
  un_current_theta0 = un_new_theta0
  un_current_theta1 = un_new_theta1
}
```

Q2

```r
# cbind binds columns
# rep(1, number of times to repeat)
xnorm <- cbind(rep(1,nrow(housing_scaled)), housing_scaled$size)

thetahat <- solve((t(xnorm)%*%(xnorm)))%*%t(xnorm)%*%(y)
print(thetahat)
```

```
##          [,1]
## [1,] 340412.7
## [2,] 106907.6
```

Q3

```
logL = function(thetas, m, x, y){
  # note that thetas is a vector #
  sigma=1 # we assume sigma=1
  m*log(1/(sqrt(2*pi)*sigma)) - 1/(sigma^2)*(1/2)*sum((thetas[1] + thetas[2]*x - y)^2)
}

logL(c(4,5),m,x,y)
```

```
## [1] -3.082714e+12
```

```
#what is thetas?
thetas <- c(theta0_values,theta1_values)

mle <- optim(par=(c(0,0)), logL, x=x, y=y, m=m, method = "L-BFGS-B",control = list(fnscale=-1))
print(mle$par)
```

```
## [1] 340412.7 106907.6
```