Stat 432 Homework 3

Giang Le (gianghl2)

Assigned: Sep 6, 2021; Due: 11:59 PM CT, Sep 14, 2021

Contents

Question 1: Optimizing a Mean Model

For this question, you need to write functions to iteratively update the parameter to reduce the functional value, meaning that your function must contain the iteration steps. You can still use other build-in R functions to simplify the calculation. You cannot use the `optim()` function unless explicitly asked to.

The goal of this question is to estimate the mean of a set of samples using numerical optimization. In other words, we observe $\{y_i\}_{i=1}^n$ and using the ℓ_2 loss, which is the same as a regression model, we want to estimate θ by minimizing the objective function

$$\ell(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \theta)^2$$

Generate 100 observations using the following code, change the seed to you UIN.

```
set.seed(662095561)
y = rnorm(100)
mean(y)
```

```
## [1] -0.1366709
```

a) [10 Points] Write a function `mean_loss(theta, trainy)` that calculates the loss function given an θ values and a vector **y** of observations.

My function takes a single theta value and a vector of y values. It computes the mean loss function over the y observations based on the formula

$$\ell(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \theta)^2$$

```
mean_loss <- function(theta, trainy) {
  return(mean((trainy - theta)^2))
}
```

b) [10 Points] Use your function to calculate its value at a grid of θ values using `seq(-1.5, 1.5, 0.01)`. Plot the objective function in a figure, with θ as the horizontal axis and the objective function value as the vertical axis. The figure may look similar to this one. Add the optimal point to the figure.

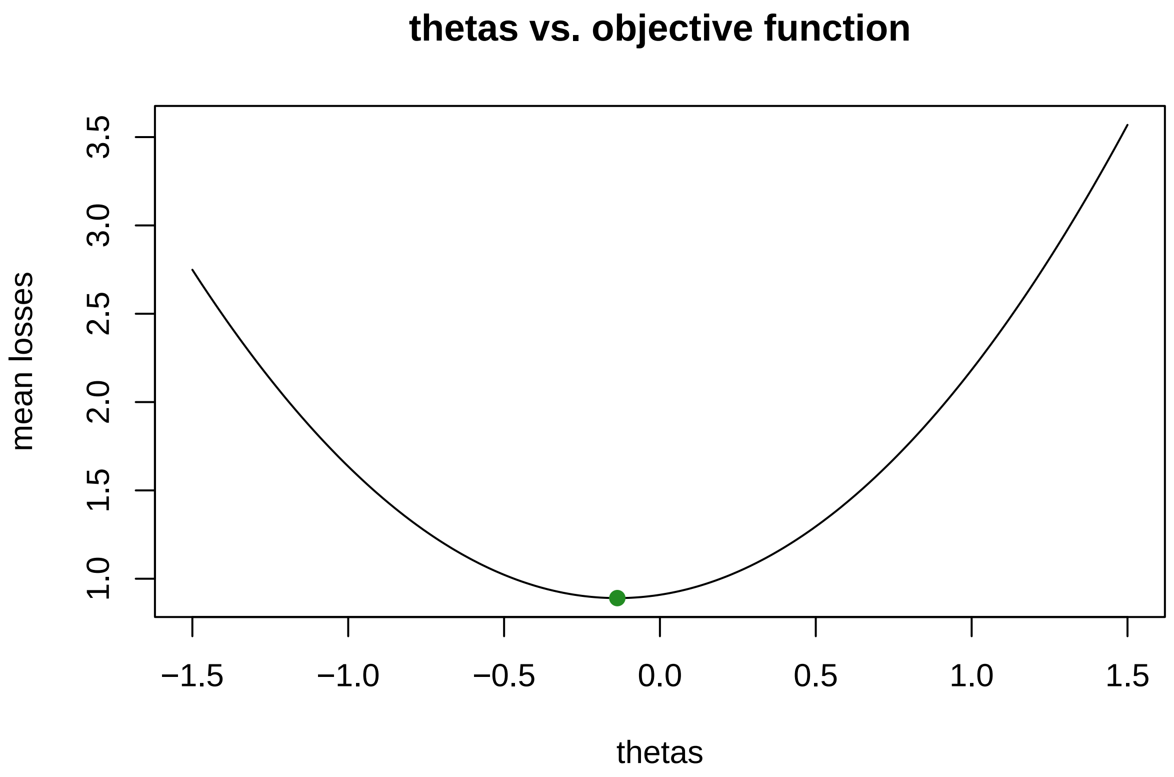
Below I created a sequence of theta values and then I looped through this seq to get the mean_loss result from each theta compared to the train y values.

```r
thetas <- seq(-1.5, 1.5, 0.01)
results <-rep(0, length(thetas))
for (i in 1:length(thetas)) {
  results[i] <- mean_loss(thetas[i], trainy=y)
}
```

```r
theta_optimal <- rep(mean(y),1)
data <- data.frame(thetas, results)
library(ggplot2)
originalplot <- ggplot(data, aes(x=thetas, y=results)) +
  geom_point(colour = "blue", size = 0.5) +
    geom_point(aes(x=mean(y), y=mean_loss(theta_optimal, trainy=y)), colour = "red", size = 1.0) +
  labs(x="theta values", y="mean losses")
```

I plotted the theta vs. mean losses and chose the lowest mean loss point as the optimal point. The optimal point is where $\hat{\theta} = \bar{y}$, which I will show in the later part of this assignment as the point where the mean loss is minimized.

```r
plot(x=thetas, y=results, main = "thetas vs. objective function", type="l",
     cex=0.5, xlab="thetas", ylab="mean losses")
points(mean(y), mean_loss(theta_optimal, trainy=y) , pch = 19, col = "forest green")
```



~~~
  c) [10 Points] Test this function using the `optim()` function and obtain the optimizer, which should

I called the optim() function to obtain the optimal value for theta. We can see that the optimizer found

```r
optim(par=1, fn = mean_loss, method="BFGS", trainy=y)
~~~

```
## $par
## [1] -0.1366709
```

```
##
## $value
## [1] 0.8903873
##
## $counts
## function gradient
##        6        3
##
## $convergence
## [1] 0
##
## $message
## NULL
```

d) Write your own gradient descent algorithm to solve this problem. To do this, complete the following steps:
   i) [10 Points] Derive the gradient (derivative with respect to θ) of the objective function, and write that down using Latex

I took the derivative with respect to θ and show that the optimal value of θ equals to the sample mean or $\bar{y}$

$$\frac{\partial l(\theta)}{\partial \theta} = \frac{-2}{n} \sum_{i=1}^{n} (y_i - \theta) \tag{1}$$

$$\frac{\partial l(\theta)}{\partial \theta} = -2\bar{y} + 2\theta \tag{2}$$

Setting this derivative to 0 means finding theta such that:

$$\sum_{i=1}^{n} (y_i - \theta) = 0 \tag{3}$$

$$\sum_{i=1}^{n} y_i - n\theta = 0 \tag{4}$$

$$\hat{\theta} = \frac{\sum_{i=1}^{n} y_i}{n} \tag{5}$$

$$\hat{\theta} = \bar{y} \tag{6}$$

ii) [5 Points] Perform a calculation of this gradient at $\theta = 1$. The result should be positive because the objective function should be moving upwards at $\theta = 1$. However, note that during the parameter update, you should move towards the negative gradient.

At $\theta = 1$ and the given y values, the gradient is shown below.

```r
-2*mean(y) + 2
```

```
## [1] 2.273342
```

```
iii) [25 Points] Write your own function `optim_mean_g(trainy, theta0,
delta, epsilon, maxitr)` to solve for the optimizer.
If you need an example of this, see the [First-order Methods](https://teazrq.github.io/stat432/RNotes
      * `theta0` is the initial value
      * `delta` is the step size, with default value 0.3.
```

```
        * `epsilon` is the stopping rule criteria
        * `maxitr` is the maximum number of iterations
```

I wrote the function optim_mean_g. It involves initializing the parameter value theta, update theta iteratively based on the first-order method. The algorithm stops when the difference between the new parameter value and the old parameter value smaller than some epsilon value or when the max iteration is reached. All parameter values are saved into a vector, and the optimal value is also returned by the function.

```r
optim_mean_g <- function(trainy,
                         theta0, # the initial value
                         delta=0.3, # step size, default 0.3
                         epsilon, # stopping rule criteria
                         maxitr) {

  # initialize theta values
  alltheta = rep(theta0, maxitr)

  # iterative update
  for (k in 1:maxitr) {

  # update theta
  theta1 = theta0 - delta*(-2*mean(trainy) + 2*theta0)

  # record the new theta
  alltheta = rbind(alltheta, as.vector(theta1))

  # stopping rule
  if (max(abs(theta0 - theta1)) < epsilon)
    break;

  # reset theta0
  theta0 = theta1
  }

  if (k == maxitr) cat("maximum number of iterations reached\n")
    return(list("alltheta" = c(alltheta), "optimal" = theta1))

}
```

```
  iv) [10 Points] Finally, run your algorithm with initial value
  $\theta_0 = 1$ and report the optimizer.
```

The optimizer is

```r
optim_mean_g(trainy=y, theta0=1, delta=0.3, epsilon=0.001, maxitr=20)$optimal
```

```
## [1] -0.1363729
```

Difference between this optimizer and the sample mean is

```r
optim_mean_g(trainy=y, theta0=1, delta=0.3, epsilon=0.001, maxitr=20)$optimal - mean(y)
```

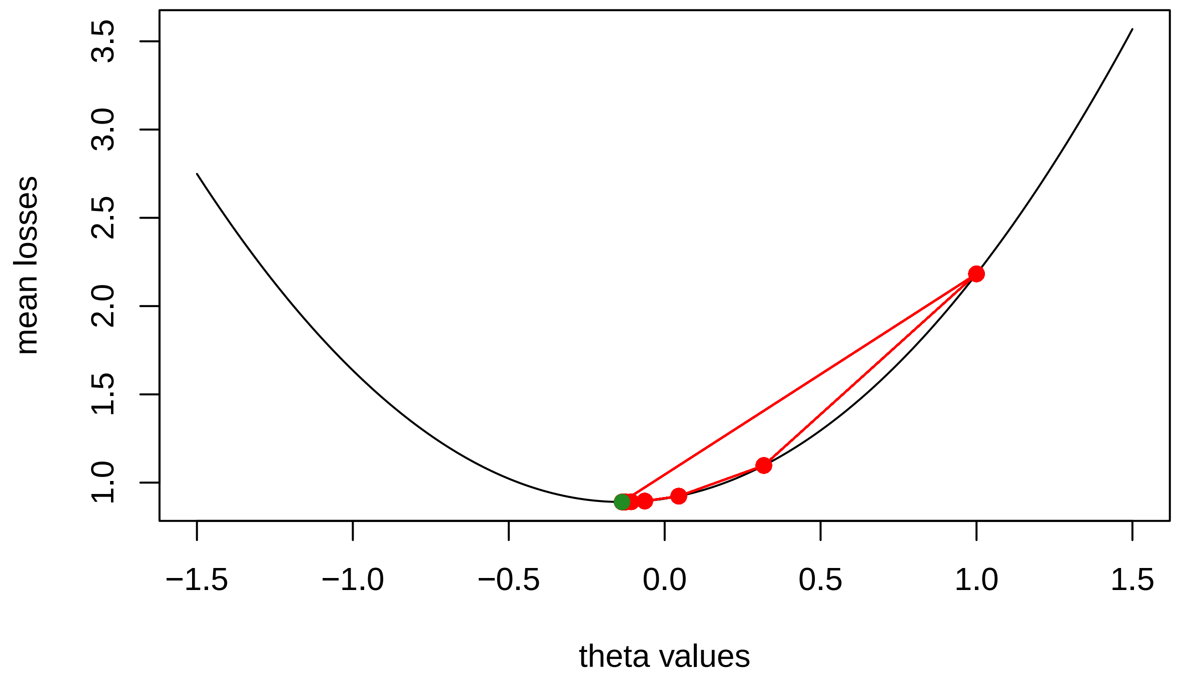
```
## [1] 0.0002979715
```

So the algorithmn is able to report a value for theta that is 0.000298 from the sample mean after only 20 iterations.

```
  v) [10 Points] Plot the path of your gradient descent in the figure you constructed previously.
  Choose step size = $0.9$. What do you observe?
  Comment on the difference between these two situations and the impact of the step size. The following
```

Here is the path of gradient descent when using a step size of 0.3

```r
thetapath1 <- optim_mean_g(trainy=y, theta0=1, delta=0.3, epsilon=0.001, maxitr=20)$alltheta
losses <- rep(0, length(thetapath1))
for (i in 1:length(thetapath1)) {
  losses[i] <- mean_loss(thetapath1[i], trainy=y)
}
plot(thetas, results, xlab="theta values", ylab="mean losses", type="l", cex=0.5)
for (i in 1:length(thetapath1)) {
  points(thetapath1[i], losses[i], type = "b", col = "red", pch = 19)
}
lines(thetapath1, losses, type="l", col = "red")
points(mean(y), mean_loss(theta_optimal, trainy=y) , pch = 19, col = "forest green")
```

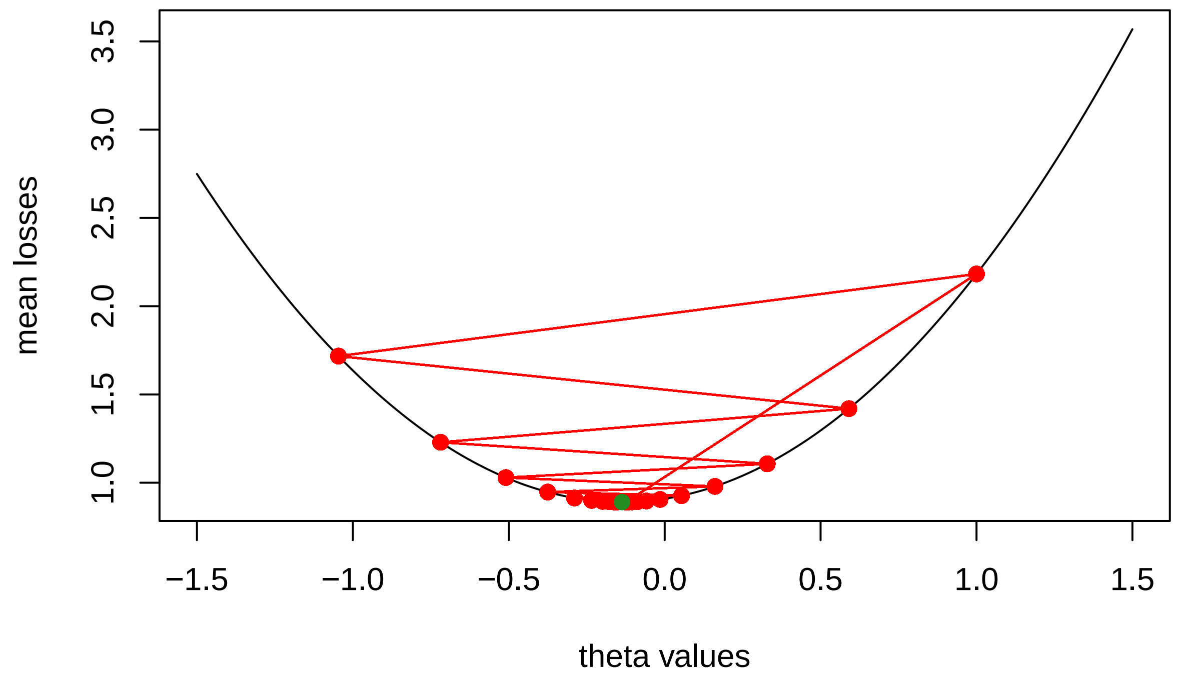


Here is the path of gradient descent when using a step size of 0.9

```r
thetapath2 <- optim_mean_g(trainy=y, theta0=1, delta=0.9, epsilon=0.001, maxitr=20)$alltheta
```

```
## maximum number of iterations reached
```

```r
losses <- rep(0, length(thetapath2))
for (i in 1:length(thetapath2)) {
  losses[i] <- mean_loss(thetapath2[i], trainy=y)
}
plot(thetas, results, xlab="theta values", ylab="mean losses", type="l", cex=0.5)
for (i in 1:length(thetapath2)) {
  points(thetapath2[i], losses[i], type = "b", col = "red", pch = 19)
}
lines(thetapath2, losses, type="l", col = "red")
points(mean(y), mean_loss(theta_optimal, trainy=y) , pch = 19, col = "forest green")
```



The difference between the optimizer and the sample mean when the step size is 0.9

```
optim_mean_g(trainy=y, theta0=1, delta=0.9, epsilon=0.001, maxitr=20)$optimal - mean(y)
```

```
## maximum number of iterations reached
```

```
## [1] 0.01310492
```

The thetapaths are shown below. We can see that the step size of 0.9 returns a theta value that is further away from the sample mean after 20 iterations compared to the step size of 0.3 (which converged before 20 iterations is reached). So the step size of 0.3 results in a more efficient search and a step size of 0.9 is too big and results in overshooting the target.