HW3 Q7,8

Muhammad Ahmed Chaudhry

5/31/2021

Question 7

a) Here we code the function to implement SGD with mini-batches:

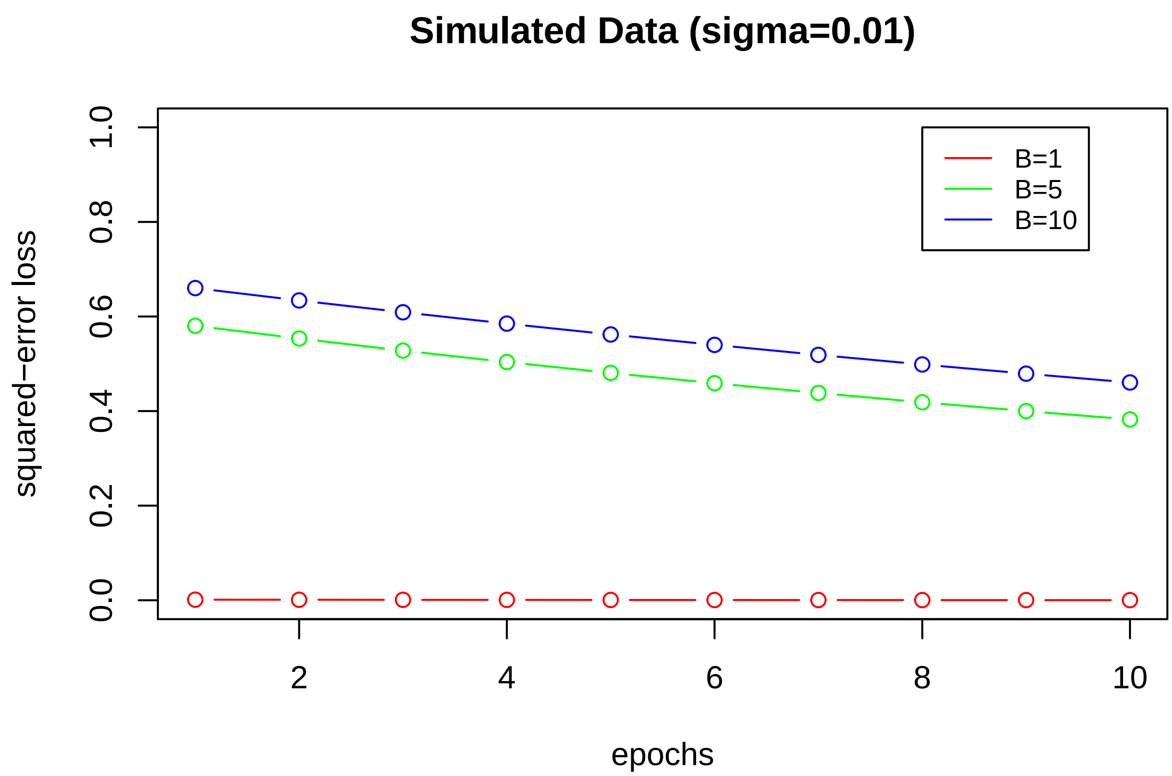
```
stochastic_gd <- function(X,y,epsilon,batch_size,epochs){
  #' Function written as per required inputs and outputs
  #' @param X: matrix, data-set of input variable values of size N*n
  #' @param y: vector of responses for each given set of input variable values, size N*1
  #' @param epsilon: numeric, value for the learning rate at each step
  #' @param B: batch size, number of observations to use at each step when computing gradient
  #' @param epochs: integer (positive), number of passes through the entire data-set
  #' @return squared_loss: matrix, of squared error loss values, one for each step, size epochs*N/B

  B<-batch_size
  #print(B)
  N<-dim(X)[1]
  #print(N)
  n<-dim(X)[2]
  #print(n)
  #indices<-seq(1,N/B,length.out=N)
  #print(indices)
  beta<-matrix(0,nrow=n,ncol=1)
  squared_loss<-matrix(NA,nrow=epochs,ncol=N/B)
  # loss_0<-1/N*t(y)%*%y
  # squared_loss[1,]<-rep(loss_0,N/B)

  for (j in 1:epochs){
    loss<-c(rep(0,N/B))
    for (i in 1:(N/B)){
      start<-(i-1)*B+1
      #print(start)
      end<-start+B-1
      #print(end)
      predictors<-as.matrix(X[start:end,])
      response<-y[start:end]
      if (B==1){
        gradient<-2/N*(as.matrix(predictors)%*%(response-t(as.matrix(predictors))%*%beta))
        #print(gradient)
        #print(dim(gradient))
        beta<-beta+epsilon*gradient
        #print(beta)
        #print(dim(beta))
```

```
        loss_iter<-1/N*(response-t(as.matrix(predictors))%*%beta)^2
        #print(dim(loss_iter))
        loss[i]<-loss_iter
      }
      else{
        gradient<-2/N*(t(as.matrix(predictors))%*%(response-as.matrix(predictors)%*%beta))
        #print(gradient)
        #print(dim(gradient))
        beta<-beta+epsilon*gradient
        #print(beta)
        #print(dim(beta))
        loss_iter<-1/N*(t(response-as.matrix(predictors)%*%beta)%*%(response-as.matrix(predictors)%*%bet
        #print(dim(loss_iter))
        loss[i]<-loss_iter
      }
    }
    squared_loss[j,]<-loss
  #print(loss)
  }
  #print(squared_loss)



return(squared_loss)
}
```

Here we generate the data to test the function

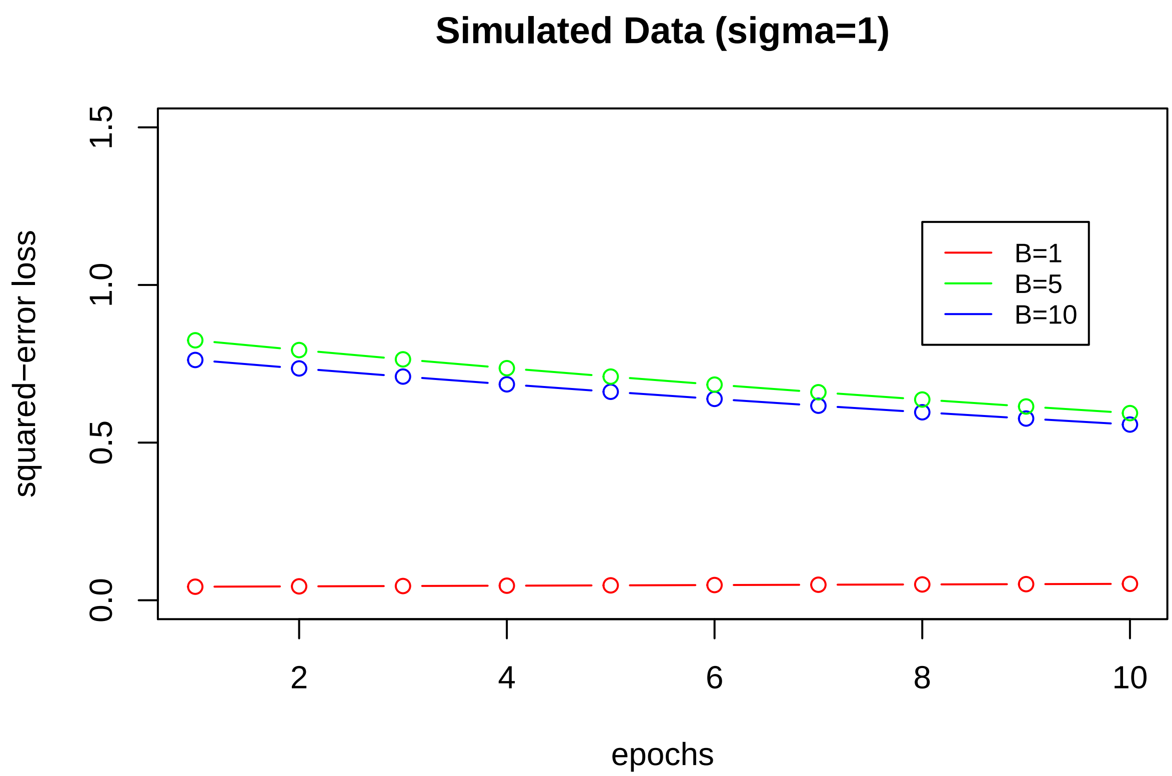
```
set.seed(2021)
sigma<-0.01
X<-mvrnorm(n=100, mu=rep(0,10),Sigma=diag(rep(1,10)))
a_star<-mvrnorm(n=1, mu=rep(0,10),Sigma=diag(rep(1,10)))
delta<-mvrnorm(n=1, mu=rep(0,100),Sigma=diag(rep(sigma^2,100)))

y<-X%*%a_star+delta


result_1<-stochastic_gd(X,y,0.01,1,10)
result_5<-stochastic_gd(X,y,0.01,5,10)
result_10<-stochastic_gd(X,y,0.01,10,10)
```

```
epochs<-1:10
loss_epochs_1<-result_1[,dim(result_1)[2]]
loss_epochs_5<-result_5[,dim(result_5)[2]]
loss_epochs_10<-result_5[,dim(result_10)[2]]




plot(x=epochs,y=loss_epochs_1,type="b",col="red",ylim=(c(0,1)),ylab="squared-error loss",main="Simulated
points(x=epochs,y=loss_epochs_5,type="b",col="blue")
points(x=epochs,y=loss_epochs_10,type="b",col="green")
legend(8,1, legend=c("B=1", "B=5", "B=10"),
       col=c("red", "green", "blue"), lty=1, cex=0.8)
```



Here we change the value of sigma

```
set.seed(2021)
sigma<-1
X<-mvrnorm(n=100, mu=rep(0,10),Sigma=diag(rep(1,10)))
a_star<-mvrnorm(n=1, mu=rep(0,10),Sigma=diag(rep(1,10)))
delta<-mvrnorm(n=1, mu=rep(0,100),Sigma=diag(rep(sigma^2,100)))

y<-X%*%a_star+delta


result_1<-stochastic_gd(X,y,0.01,1,10)
result_5<-stochastic_gd(X,y,0.01,5,10)
result_10<-stochastic_gd(X,y,0.01,10,10)


epochs<-1:10
loss_epochs_1<-result_1[,dim(result_1)[2]]
loss_epochs_5<-result_5[,dim(result_5)[2]]
loss_epochs_10<-result_5[,dim(result_10)[2]]


plot(x=epochs,y=loss_epochs_1,type="b",col="red",ylim=c(0,1.5),ylab="squared-error loss", main="Simulat
points(x=epochs,y=loss_epochs_5,type="b",col="blue")
points(x=epochs,y=loss_epochs_10,type="b",col="green")
legend(8,1.2, legend=c("B=1", "B=5", "B=10"),
       col=c("red", "green", "blue"), lty=1, cex=0.8)
```



Simulated Data (sigma=1)
squared−error loss
0.0
0.5
1.0
1.5
2
4
6
8
10
epochs
B=1
B=5
B=10

Question 8