Problem 2

Xinyi Song

10/14/2020

Problem 2: Using the dual nature to our advantage

Sometimes using a mixture of true matrix math plus component operations cleans up our code giving better readibility. Suppose we wanted to form the following computation:

- $while(abs(\Theta_0^i - \Theta_0^{i-1})$ AND $abs(\Theta_1^i - \Theta_1^{i-1}) > tolerance)$ {

$$\begin{aligned}\Theta_0^i &= \Theta_0^{i-1} - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_0(x_i) - y_i) \\ \Theta_1^i &= \Theta_1^{i-1} - \alpha\frac{1}{m}\sum_{i=1}^{m}((h_0(x_i) - y_i)x_i)\end{aligned}$$

}

Where $h_0(x) = \Theta_0 + \Theta_1 x$.

Given $\mathbf{X}$ and $\vec{h}$ below, implement the above algorithm and compare the results with lm(h~0+$\mathbf{X}$). State the tolerance used and the step size, α.

```
set.seed(1256)
theta <- as.matrix(c(1,2),nrow=2)
X <- cbind(1,rep(1:10,10))
h <- as.vector(X%*%theta+rnorm(100,0,0.2))
m <- dim(X)[1]
#theta = matrix(0,2,1)
THETA = matrix(5,2,1)
alpha = 0.01
while((abs(theta[1] - THETA[1])>1e-06) || (abs(theta[2] - THETA[2])>1e-06)){
  THETA = theta
  h_0 = X%*%THETA
  h_y = sweep(as.matrix(h_0), 1, h, '-')
  theta[1] = THETA[1] - alpha*mean(h_y)
  h_yx = sweep(h_y, 1, as.matrix(X[,2]), '*')
  theta[2] = THETA[2] - alpha*mean(h_yx)
}
print(theta)
```

```
##           [,1]
## [1,] 0.9700454
## [2,] 2.0014948
```

```
# regression
dat = as.data.frame(cbind(h,X))
```

```r
fit = lm(h~0+X, data = dat)
summary(fit)
```

```
##
## Call:
## lm(formula = h ~ 0 + X, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.5724 -0.1342 -0.0164  0.1179  0.4807
##
## Coefficients:
##    Estimate Std. Error t value Pr(>|t|)
## X1 0.969571   0.042344    22.9   <2e-16 ***
## X2 2.001563   0.006824   293.3   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.196 on 98 degrees of freedom
## Multiple R-squared:  0.9998, Adjusted R-squared:  0.9998
## F-statistic: 2.297e+05 on 2 and 98 DF,  p-value: < 2.2e-16
```

Here, I use alpha - the step size as 0.001 and tolerance value as 0.001, the results are very closed.

Problem 3

The above algorithm is called Gradient Descent. This algorithm, like Newton's method, has "hyperparameters" that are determined outside the algorithm and there are no set rules for determing what settings to use. For gradient descent, you need to set a start value, a step size and tolerance.

Part a. Using a step size of $1e^{-7}$ and tolerance of $1e^{-9}$, try 10000 different combinations of start values for β_0 and β_1 across the range of possible β's +/-1 from true determined in Problem 2, making sure to take advantages of parallel computing opportunities. In my try at this, I found starting close to true took 1.1M iterations, so set a stopping rule for 5M. Report the min and max number of iterations along with the starting values for those cases. Also report the average and stdev obtained across all 10000 β's.

```r
set.seed(1256)
X <- cbind(1,rep(1:10,10))
theta <- as.matrix(c(1,2),nrow=2)
h <- as.vector(X%*%theta+rnorm(100,0,0.2))
THETAs = expand.grid(seq(0, 2, length.out = 100), seq(1, 3, length.out = 100))
# function of Gradient Descent
# Modify the function to do parallel programming
grad = function(theta_start, X, h){
  theta_old_0 = 1000
  theta_old_1 = 1000
  alpha = 1e-2 # step size
  tol = 1e-05 # tolerance value
  theta_new_0 = theta_start[1]
  theta_new_1 = theta_start[2]
  i = 0 # i: iteration times
```

```
  while((abs(theta_new_0-theta_old_0)>tol) || (abs(theta_new_1 - theta_old_1)>tol)){
    theta_old_0 = theta_new_0
    theta_old_1 =  theta_new_1
    theta_h = rbind(theta_old_0, theta_old_1)
    h_0 = X%*%theta_h
    h_y = sweep(as.matrix(h_0), 1, h, '-')
    theta_new_0 = theta_old_0 - alpha*mean(h_y)
    h_yx = sweep(h_y, 1, as.matrix(X[,2]), '*')
    theta_new_1 = theta_old_1 - alpha*mean(h_yx)
    i = i + 1
    if(i>5000000) break
  }
  result = c(i, theta_new_0, theta_new_1)
  return(result)
}
```

```
library(parallel)
# A good number of clusters is the numbers of available cores minus 1.
no_cores <- detectCores() - 1
cl = makeCluster(no_cores) # not work
#cl <- parallel::makeCluster(no_cores, setup_strategy = "sequential") # work
clusterExport(cl, 'X')
clusterExport(cl, 'h')
start_time <- Sys.time()
# Here, I only try two observations
a <- parApply(cl,THETAs[1:2,], 1, grad, X, h)
stopCluster(cl)
end_time <- Sys.time()
end_time - start_time
```

```
## Time difference of 0.5931869 secs
```

```
min_iteration = min(a[1,])
max_iteration = max(a[1,])
mean_theta_0 = mean(a[2,])
sd_theta_0 = sqrt(var(a[2,]))
mean_theta_1 = mean(a[3,])
sd_theta_1 = sqrt(var(a[3,]))
iteration_summary = cbind(min_iteration, max_iteration)
print(iteration_summary)
```

```
##      min_iteration max_iteration
## [1,]          2433          2444
```

```
theta_summary = cbind(mean_theta_0, mean_theta_1, sd_theta_0, sd_theta_1)
print(theta_summary)
```

```
##      mean_theta_0 mean_theta_1   sd_theta_0   sd_theta_1
## [1,]    0.9648225     2.002245 5.536107e-06 7.952095e-07
```