## STAT LEARNING 2  
## HOMEWORK 4, Problem 5

# Read in necessary libraries  
library(ggplot2)  
library(magrittr)  
library(R.matlab)  
library(geometry)  
library(dplyr)  
  
  
# Set the working directory  
setwd("C:/Users/jrdha/OneDrive/Desktop/USU\_Fa2018/Moon\_\_SLDM2/hw4/Problem5")  
  
  
# This function kicks off the recursive subGradient function.  
# It takes as arguments: max\_Iter (max number of iterations), predictors  
# (dataframe of input features), response (dataframe containing values of  
# responses), lambda (used for calculation of regularization term).  
# The function returns optimized theta vector (weights, b).  
initSubGradient <- function(max\_Iter, predictors, response, lambda){  
 b = 1  
 ws = rep(0, ncol(predictors) - 1)  
 inittheta <- c(b, ws)  
   
 # Dataframe to store info from iterations  
 iteration\_Info <- data.frame(numItr = double(0), objFun = double(0),   
 w1 = double(0), w2 = double(0), b = double(0))  
   
 # call subGradient function  
 theta <- subGradient(theta = inittheta, num\_Iter = 1, max\_Iter = max\_Iter,   
 xs = predictors, y = response,  
 lambda = lambda, iteration\_Info = iteration\_Info)  
 return(theta)  
}  
  
  
# This function (recursively) performs the calculations to determine optimal value  
# of theta or calls itself again.  
# the initSubGradient function.  
# Takes as arguments: theta (vector of weights and b), num\_Iter and max\_Iter (are  
# self-explanatory), xs (dataframe containing input data), y (dataframe containing  
# response data), lambda (regularization parameter), iteration\_Info (dataframe  
# that stores parameter estimates and value of objective function).  
subGradient <- function(theta, num\_Iter, max\_Iter, xs, y, lambda, iteration\_Info){  
   
 # Alpha is the step size, n is the number of instances in preds  
 alpha <- 100/num\_Iter  
 n <- nrow(preds)  
   
 # Either retunr optimized theta and iteration info, or keep going  
 if(num\_Iter > max\_Iter){  
 return(list("theta" = theta, "iteration\_Info" = iteration\_Info))  
 } else {  
 print(paste("Iteration: ", num\_Iter))  
   
 u <- double(length(theta))  
 for(i in 1:nrow(xs)){  
 # calculate gradient  
 grad <- calcGrad(b = theta[1], w = theta[2:length(theta)],   
 yi = as.numeric(y$Y[i]), xi = as.numeric(xs[i,]),   
 n = n, lambda = lambda)  
 # Update u  
 u <- u + grad  
 }  
   
 # Having calculated gradient, take a step in the opposite direction  
 theta.1 <- theta - (alpha \* u)  
   
 # calculate value of objective function  
 objFctn\_val <- calc\_objFctn(preds = xs, resp = y, theta = theta.1, lambda = lambda)  
   
 # This row will be added to the bottom of the iteration\_Info dataframe  
 hnew <- data.frame(numItr = num\_Iter, objFun = objFctn\_val, b = theta.1[1],   
 w1 = theta.1[2], w2 = theta.1[3])  
   
 iteration\_Info <- rbind(iteration\_Info, hnew)  
 num\_Iter <- num\_Iter + 1  
   
 return(subGradient(theta = theta.1, num\_Iter = num\_Iter, max\_Iter = max\_Iter,   
 xs = xs, y = y, lambda = lambda, iteration\_Info = iteration\_Info))  
 }  
}  
  
  
# This function kicks of the recursive stochSubGradient function.  
# It takes as arguments: max\_Iter, predictors, response, and lambda (we've seen  
# all of these before, so I won't re-explain what they are).  
# It returns the optimal value of the theta vector.  
initStochSubGradient <- function(max\_Iter, predictors, response, lambda){  
   
 # Initial guesses for theta (b and the weights)  
 b = 1  
 ws = rep(0, ncol(predictors) - 1)   
 inittheta <- c(b, ws)  
   
 # Initialize the iteration\_Info data frame  
 iteration\_Info <- data.frame(numItr = double(0), objFun = double(0),   
 w1 = double(0), w2 = double(0), b = double(0))  
   
 # Recursion of stochSubGradient() begins  
 theta <- stochSubGradient(theta = inittheta, num\_Iter = 1, max\_Iter = max\_Iter,   
 xs = predictors, y = response,  
 lambda = lambda, iteration\_Info = iteration\_Info)  
 return(theta)  
}  
  
  
# This recursive function calls the subGradient function.   
# Takes as arguments all the things we know and love: theta vector, num\_Iter,  
# max\_Iter, xs, y, lambda, and iteration\_Info dataframe.  
# This function either returns the optimal value of theta, or calls itself again  
# depending on how many iterations have taken place.  
stochSubGradient <- function(theta, num\_Iter, max\_Iter, xs, y, lambda, iteration\_Info){  
   
 # Set the step size, how many predictor instances there are, and recommended  
 # minibatch size  
 alpha <- 100/num\_Iter  
 n <- nrow(preds)  
 m <- 1  
   
 # The use of the sample() function below will be how we randomly sample  
 if(num\_Iter > max\_Iter){  
 return(list("theta" = theta, "iteration\_Info" = iteration\_Info))  
 } else {  
 print(paste("Iteration Number: ", num\_Iter))  
 index <- 1:n  
 rand.index <- sample(index, n)  
 for(i in rand.index){  
 grad <- calcGrad(b = theta[1], w = theta[2:length(theta)],  
 yi = as.numeric(y$Y[i]), xi = as.numeric(xs[i,]),  
 n = 1, lambda = lambda)  
 theta <- theta - (alpha \* grad) # Move in opposite direction of subGrad  
 # to update theta.  
 }  
 theta.1 <- theta  
   
 # Calculate the (current) value of the objective function.  
 objFctn\_val <- calc\_objFctn(preds = xs, resp = y, theta = theta.1, lambda = lambda)  
   
 # This row will be added to the bottom of the iteration\_Info dataframe  
 hnew <- data.frame(numItr = num\_Iter, objFun = objFctn\_val, b = theta.1[1],   
 w1 = theta.1[2], w2 = theta.1[3])  
   
 iteration\_Info <- rbind(iteration\_Info, hnew)  
 num\_Iter <- num\_Iter + 1  
 return(subGradient(theta = theta.1, num\_Iter = num\_Iter, max\_Iter = max\_Iter, xs = xs, y = y, lambda = lambda, iteration\_Info = iteration\_Info))  
 }  
}  
  
  
# This function calculates the gradient.  
# Takes as arguments: w (weights vector), b (part of theta, intercept term),   
# yi (response), xi (vector containing predictor instance), n (num of instances),  
# lambda (same parameter for calculating regularization penalty term).  
# Returns the subGradient.  
calcGrad <- function(w, b, yi, xi, n, lambda){  
  
 wvec <- c(0, w)  
 term <- 1 - (yi \* (dot(wvec, xi) + b))  
 if(term >= 0){  
 gradJi <- (1/n)\*(-yi\*xi) + (lambda/n)\*wvec  
 } else {  
 gradJi <- (lambda/n)\*wvec  
 }  
 sg <- (gradJi)  
 return(sg)  
}  
  
  
# Calculates the value of the objective function for given inputs.  
# Takes as arguments: preds (predictor data), resp (response value), theta (same  
# vector of weights and b), lambad (same regularization penalty parameter).  
calc\_objFctn <- function(preds, resp, theta, lambda){  
  
 n <- nrow(preds)  
 b <- theta[1]  
 w <- theta[2:length(theta)]  
 wvec <- c(0, w)  
 val\_Summation <- 0  
 for(i in 1:n){  
 yi = as.numeric(resp$Y[i])  
 xi = as.numeric(preds[i,])  
 m1 <- 0  
 m2 <- 1 - yi\*(dot(wvec, xi) + b)  
 term <- max(m1, m2)  
 val\_Summation <- val\_Summation + term  
 }  
 val\_objFctn <- 1/n \* (val\_Summation) + (lambda/2)\*dot(w,w)  
 return(val\_objFctn)  
}  
  
# This function plots the values of the objective functions against the iteration  
# number for that given value of J.  
# Takes as arguments the iteration\_Info dataframe for plotting, and a title.  
# Output is the plot described above.  
plot\_objFctn\_vals <- function(iteration\_Info, title){  
 plot(iteration\_Info$numItr, iteration\_Info$objFun, xlab = "Iteration Number",   
 ylab = "value of Objective Function J", type = 'o',  
 main = title)  
}  
  
  
#===============================================================================  
#== Implementing the defined functions to solve the given problem ==============  
#===============================================================================  
  
givenData <- R.matlab::readMat("nuclear.mat") %>% lapply(t) %>% lapply(as\_tibble)  
colnames(givenData[[1]]) <- sprintf("X\_%s",seq(1:ncol(givenData[[1]])))  
colnames(givenData[[2]]) <- c("Y")  
givenData <- bind\_cols(givenData) %>% select(Y, everything())  
  
# givenData <- slice(givenData, 1:300)  
preds <- select(givenData, X\_1, X\_2)  
X0 <- rep(1, nrow(givenData))  
preds <- cbind(X0, preds)  
resp <- select(givenData, Y)  
resp$Y <- as.numeric(resp$Y)  
  
  
#=== Results of implementing subGradient method ================================  
  
# Seems to flatten out after about 35 iterations, chose a fairly small value of lambda.  
subGrad\_results <- initSubGradient(max\_Iter = 35, predictors = preds, response = resp, lambda = 0.001)

plot\_objFctn\_vals(subGrad\_results$iteration\_Info, title = "SubGradient Descent Method Results")

b <- subGrad\_results$theta[1]  
w1 <- subGrad\_results$theta[2]  
w2 <- subGrad\_results$theta[3]  
  
slopeVal <- w1/-w2  
interceptVal <- b/-w2  
  
# Gets mad if not converted to the factor data type  
givenData$Y <- as.factor(givenData$Y)  
  
ggplot(data = givenData, aes(x = X\_1, y = X\_2, color = Y)) +   
 geom\_point() +   
 scale\_color\_manual(values=c("-1" = "turquoise", "1" = "magenta")) +   
 geom\_abline(slope = slopeVal, intercept = interceptVal, lwd = 1) +  
 xlab("Values of X1") +  
 ylab("Values of X2") +   
 ggtitle("Separation using SubGradient Method")

#=== Results of implementing stochSubGradient method ===========================  
# This one took a lot more iterations to converge as compared to subGradient method.  
stoch\_subGrad\_results <- initStochSubGradient(max\_Iter = 100, predictors = preds,   
 response = resp, lambda = 0.001)

plot\_objFctn\_vals(stoch\_subGrad\_results$iteration\_Info, title = "Stochastic SubGradient Descent Method Results")

b <- stoch\_subGrad\_results$theta[1]  
w1 <- stoch\_subGrad\_results$theta[2]  
w2 <- stoch\_subGrad\_results$theta[3]  
  
slopeVal <- w1/-w2  
interceptVal <- b/-w2  
  
plot(preds$X\_1, preds$X\_2)  
abline(a = interceptVal, b = slopeVal)

# Again, have to change this data type to factor or it'll get mad  
givenData$Y <- as.factor(givenData$Y)  
  
ggplot(data = givenData, aes(x = X\_1, y = X\_2, color = Y)) +   
 geom\_point() +   
 scale\_color\_manual(values=c("-1" = "turquoise", "1" = "magenta")) +   
 geom\_abline(slope = slopeVal, intercept = interceptVal, lwd = 1) +  
 xlab("Values of X1") +  
 ylab("Values of X2") +   
 ggtitle("Separation using Stochastic SubGradient Method")