hw6

Qiuying Li UNI ql2280

4/27/2017

Problem 1 (Implementing PageRank, 20 points)

setwd("~/Desktop/2017 spring/GR 5241/HW/hw6")  
graph = as.matrix(read.table("graph.txt", header=FALSE ))  
num\_node = 100  
alpha = 0.2  
A=matrix(0,num\_node,num\_node)  
for (i in 1:100){  
 destination=graph[graph[,1]==i,]  
 A[destination[,2],i]=1/nrow(destination)  
}  
r = rep(1/num\_node, num\_node)  
for(iter in 1:40){  
 one\_vector = rep(1, num\_node)  
 r = alpha / num\_node \* one\_vector + (1 - alpha) \* A %\*% r  
}  
#List the top 5 nodes ids with the highest PageRank scores (submit both ids and scores).  
top\_index = order(r)[1:5]  
print(rbind(top\_index,r[top\_index]),digits = 3)

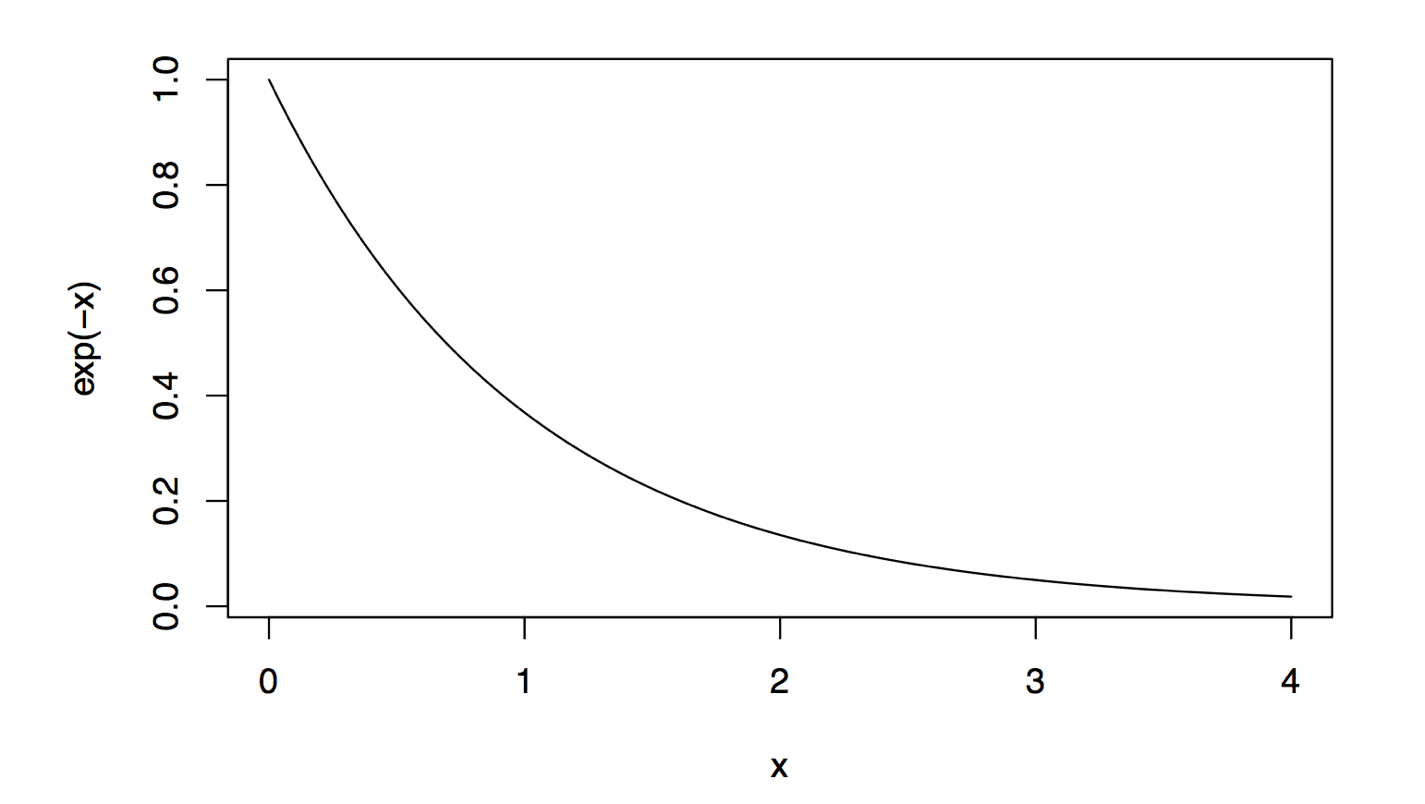
## [,1] [,2] [,3] [,4] [,5]  
## top\_index 85.0000 59.00000 81.00000 23.00000 37.00000  
## 0.0031 0.00329 0.00334 0.00337 0.00345

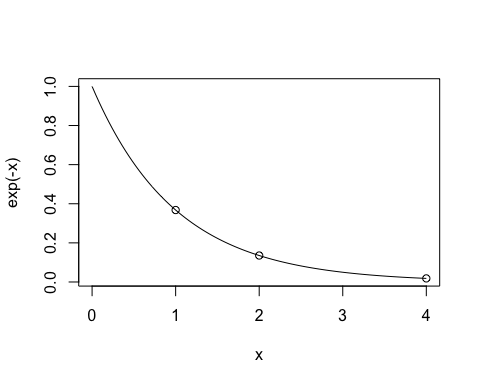
#List the bottom 5 node ids with the lowest PageRank scores (submit both ids and scores).  
bottom\_index = order(-r)[1:5]  
print(rbind(bottom\_index,r[bottom\_index]), digits = 3)

## [,1] [,2] [,3] [,4] [,5]  
## bottom\_index 53.0000 40.0000 14.0000 1.0000 27.0000  
## 0.0261 0.0251 0.0247 0.0223 0.0223

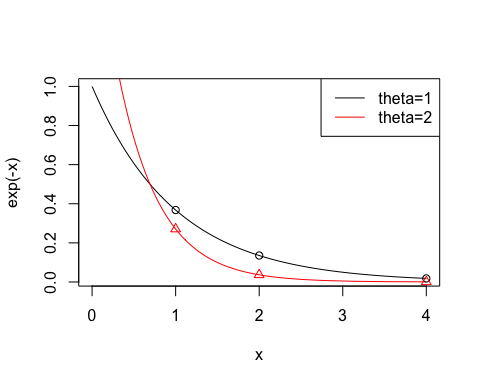
Problem 2 (Bayesian inference, 20 points)

#1. Plot the graph of p(x;θ) for θ = 1 in the interval x ∈ [0,4].  
curve(exp(-x), from=0, to=4)

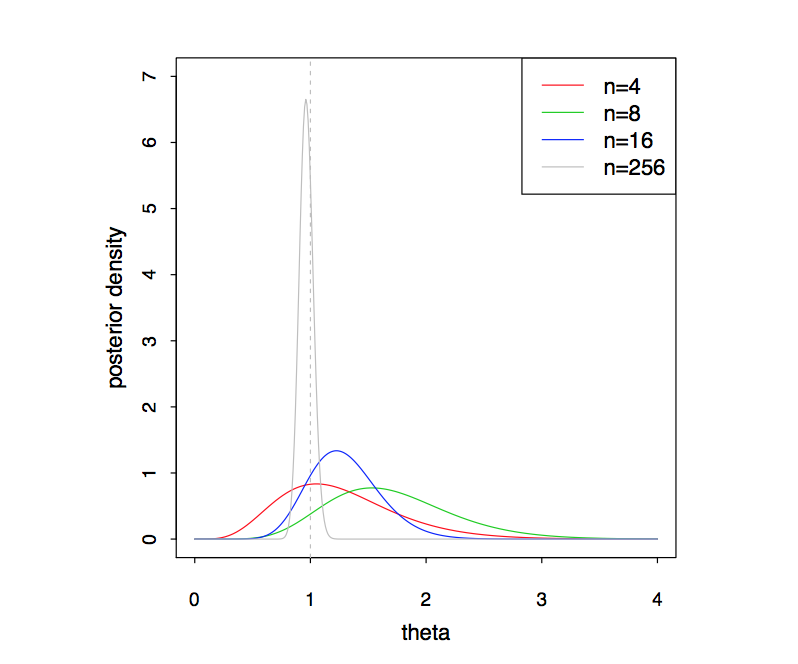
  
#2  
curve(exp(-x), from=0, to=4)   
points(1, exp(-1), pch=1)   
points(2, exp(-2), pch=1)   
points(4, exp(-4), pch=1)



curve(exp(-x), from=0, to=4)  
curve(2\*exp(-2\*x), from=0, to=4, add=T, col='red')  
points(1, exp(-1), pch=1); points(1, 2\*exp(-2), pch=2, col='red');   
points(2, exp(-2), pch=1); points(2, 2\*exp(-4), pch=2, col='red');   
points(4, exp(-4), pch=1); points(4, 2\*exp(-8), pch=2, col='red');   
legend("topright",legend=c('theta=1','theta=2'), lty=1, col=c(1,2))



#3. Visualize the gradual change of shape of the posterior Π(θ|x1:n,α0,β0) with increasing n:  
  
x = rexp(256,rate=1)   
a = 2  
b = 0.2  
plot(0,ylab="Post density",xlab="theta", type='n',xlim=c(0,4),ylim=c(0,7),axes=F )  
for (n in c(4,8,16,256)){  
 alpha=n+a  
 beta=0.2+sum(x[1:n])  
 theta=0:400/100   
 lines(theta,dgamma(theta,alpha,rate=beta),type='l',col=log2(n))  
}   
legend('topright',c('n=4','n=8','n=16','n=256'),lty=1,col=log2(c(4,8,16,256)),cex=0.6)



Problem 3 (Implementation of SVM via Gradient Descent, 6 extra points)

#Reading data

#n is the number of samples in the training data

#d is the dimensions of w

features = read.table("features.txt", header=F, sep=',')  
target = read.table("target.txt", header=F)  
features = as.matrix(features)  
target = as.matrix(y)

d = ncol(features)*;*n = nrow(features)  
Cost\_Fun = function(w,b){

# Batch Gradient Decent

#Eta is the learning rate of the gradient, in this case is 0.0000003

#The convergence criteria for below is dealta\_cost < error

Eta= 0.0000003  
error = 0.25  
k = 1

# Buidling the cost function

# function(w,b) is the value of equation at kth iteration

Cost\_Fun = function(w,b){  
 w2 = sum(w\*w)  
 wx = 0  
 for (i in 1:n)   
 wx = wx + max(0, 1- target [i]\*(sum(w\*features[i,])+b))  
 return(as.numeric(0.5\*w2+100\*wx))  
}  
  
Cost\_Fun\_1 = function(w,b,j){  
 s = 0  
 for (i in 1:n)  
 s = s-y[i]\* features [i,j]\* (y[i]\*(t(features [i,]) %\*% w +b)<1)  
 return(as.numeric(w[j]+100\*s))   
}  
  
Cost\_Fun\_2 = function(w,b){  
 s = 0  
 for (i in 1:n)  
 s = s-y[i]\* (y[i]\*(t(features[i,]) %\*% w +b)<1)  
 return(as.numeric(100\*s))  
}

#Initialize w = 0, b = 0, and compute f0(w,b)

w0 = rep(0,d)  
w1 = rep(0,d)  
b0 = 0  
cpc = 1  
Cost\_Fun\_num = rep(0,2000)  
Cost\_Fun\_num[1] = Cost\_Fun(w0,b0)

# Running Batch gradient decent: Ieterate through the entire dataset and update the parameters as follows:  
while (cpc>=error){  
 for (j in 1:d)  
 w1[j] = w0[j] - Eta\* Cost\_Fun\_1(w0,b0,j)  
 b1 = b0 - Eta\* Cost\_Fun\_2(w0,b0)  
 k = k+1  
 Cost\_Fun\_num[k] = Cost\_Fun(w1,b1)  
 cpc = abs(Cost\_Fun\_num[k-1]-Cost\_Fun\_num[k])\*100/Cost\_Fun\_num[k-1]  
 w0 = w1; b0 = b1  
}  
 T1 = k  
**###Due to we need a new parameter i,we need to modify the cost function**   
Cost\_Fun\_1\_i = function(w,b,j,i){  
 if (y[i]\*(t(x[i,]) %\*% w +b)<1)  
 return(as.numeric(w[j]+100\*(-y[i]\*x[i,j])))   
 return(as.numeric(w[j]))   
}  
  
Cost\_Fun\_2\_i = function(w,b,i){  
 if (y[i]\*(t(x[i,]) %\*% w +b)<1)  
 return(as.numeric(100\*-y[i]))  
 return(0)   
}

**# Initializa ∆cost = 0, w = 0, b = 0, and compute f0(w, b)**  
w0 = rep(0,d)  
w1 = rep(0,d)  
b0 = 0  
cpc = 0  
Cost\_Fun\_num2 = rep(0,2000)  
Cost\_Fun\_num2[1] = Cost\_Fun(w0,b0)

**#For this method, eta = 0.0001, and error = 0.001**  
Eta= 0.0001  
error = 0.001  
k = 1; i = 1

**# Stochastic Gradient Method: Go through the dataset and update the parameters, one training sample at a time, as follows:**  
while ((cpc>=error)|(START)){  
 START = FALSE  
 for (j in 1:d)  
 w1[j] = w0[j] - Eta\* Cost\_Fun\_1\_i(w0,b0,j,i)  
 b1 = b0 - Eta\* Cost\_Fun\_2\_i(w0,b0,i)  
 i = i%%n+1  
 k = k+1  
 Cost\_Fun\_num2[k] = Cost\_Fun(w1,b1)  
 cpc = 0.5\*cpc+0.5\*abs(Cost\_Fun\_num2[k-1]-Cost\_Fun\_num2[k])\*100/Cost\_Fun\_num2[k-1]  
 w0 = w1; b0 = b1  
}

T2 = k;

**# In the Mini Batch Methid, we need a new parameter l, then it is nessacery to modify the funciton**  
Cost\_Fun\_1\_l = function(w,b,j,l){  
 s = 0  
 for (i in (l\*batch\_size+1):min(n,(l+1)\*batch\_size))  
 s = s-y[i]\*features[i,j]\* (y[i]\*(t(features[i,]) %\*% w +b)<1)  
 return(as.numeric(w[j]+100\*s))   
}  
  
Cost\_Fun\_2\_l = function(w,b,l){  
 s = 0  
 for (i in (l\*batch\_size+1):min(n,(l+1)\*batch\_size))  
 s = s-y[i]\* (y[i]\*(t(features[i,]) %\*% w +b)<1)  
 return(as.numeric(100\*s))  
}

**# Initializa ∆cost = 0, w = 0, b = 0, and compute f0(w, b)**  
w0 = rep(0,d)   
w1 = rep(0,d)  
b0 = 0  
cpc = 0  
START = TRUE  
Cost\_Fun\_num3 = rep(0,1000000)  
Cost\_Fun\_num3[1] = Cost\_Fun(w0,b0)  
**#For this method, eta = 0.0001, and error = 0.001**  
Eta= 0.00001  
error = 0.01  
batch\_size = 20  
k = 1; l = 0

**# Mini Batch method: Go through the dataset in batches of predetermined size and update the parameters, one training sample at a time, as follows:**

while ((cpc>=error)|(START)){  
 START=FALSE  
 for (j in 1:d)  
 w1[j] = w0[j] - Eta\* Cost\_Fun\_1\_l(w0,b0,j,l)  
 b1 = b0 - Eta\* Cost\_Fun\_2\_l(w0,b0,l)  
 l = (l+1) %% floor((n-1+batch\_size)/batch\_size)  
 k = k+1  
 Cost\_Fun\_num3[k] = Cost\_Fun(w1,b1)  
 cpc = 0.5\*cpc+0.5\*abs(Cost\_Fun\_num3[k-1]-Cost\_Fun\_num3[k])\*100/Cost\_Fun\_num3[k-1]  
 w0 = w1; b0 = b1  
 cat(k,' ',Cost\_Fun\_num3[k],' ',cpc,'\n')  
}

T3 = k;

**### Plot the value of the cost function f(w,b) after each iteration vs. the number of iteration (k). Report the total time taken for convergence by each of the gradient descent techniques. What do you infer from the plots and the time for convergence?**

x1=0:(T1-1); y1=Cost\_Fun\_num[1:T1];  
x2=0:(T2-1); y2=Cost\_Fun\_num2[1:T2];  
x3=0:(T3-1); y3=Cost\_Fun\_num3[1:T3];  
  
plot(x1, y1, lty=1, type='l',  
 xlab="iteration times", ylab="f(w,b)",  
 xlim=c(0,1000), ylim=c(230000,600000))  
points(x2, y2, type='l', lty=2)  
points(x3, y3, type='l', lty=3)  
legend("topright",legend=c("Batch grandient descent","Stochastic grandient descent","Mini batch grandient descent"), lty=1:3)

