Sta 250 Optimization Homework 2

Chen Zihao 915490404

Problem 1. Newton-CG Method for L2-regularized Logistic Regression

Given a set of instance-label pairs $(x_i, y_i), i = 1, ..., n, x_i \in \mathbb{R}^d, y_i \in \{+1, -1\}$,L2-regularized logistic regression estimates the classification model w by solving the sollowing optimization problem:

$$\min_{w\in\mathbb{R}^d}\{\frac{1}{2}||w||_2^2 + \sum_{i=1}^{n}\log(1+\exp(-y_i w^T x_i))\} := f(w),$$

The data matrix is $X \in \mathbb{R}^{n\times d}$ is dense, where each row of X is a training sample. We test the algorithm using the "breast-cancer" dataset with $n = 44$ and $d = 7129$: the training data matrix X is stored in "X_cancer", and the training labels y is stored in "y_cancer" (the format is the same with hw1). Note that you are allowed to use any coding language or package in this homework.

(a) Derive the gradient and Hessian of $f(w)$. the objective function

$$\min_{w\in\mathbb{R}^d}\{\frac{1}{2}||w||_2^2 + \sum_{i=1}^{n}\log(1+\exp(-y_i w^T x_i))\} := f(w),$$

```
#set f(w) as a function
#X[1]=y,X[-1]=x
fw<-function(w){
  1/2*sum(w^2)+sum(apply(X,1,function(X){log(1+exp(-X[1]*X[-1]%*%w))}))
}
```

the gradient is

$$\nabla f(w) = \sum_{i=1}^{n}\frac{-y_i e^{-y_i w^T x_i}}{1+e^{-y_i w^T x_i}}x_i + w$$

```
#set f'(w) as a function
fw1<-function(w){
    #calculate f'(w)
    fw=rowSums(apply(X,1,function(X){
      e=exp(-X[1]*X[-1]%*%w)
     X[-1]*as.numeric((-X[1]*e)/(1+e))
    }))
    fw+w
}
```

the hession matrix of $f(w)$

$$\nabla^2 f(w) = \sum_{i=1}^{n}\frac{y_i^2 e^{-y_i w^T x_i}}{(1+e^{-y_i w^T x_i})^2}x_i x_i^T + I$$

(b) We plan to apply CG to solve the linear system (compute$\nabla^2 f(w)^{-1}\nabla f(w)$),where the main computation is the matrix vector product. The matrix vector product $\nabla^2 f(w^0)v$will need $O(d^2)$ time if we form the d by d matrix $\nabla^2 f(w)$ and then directly compute the matrix vector product. Explain a faster way to compute $\nabla^2 f(w)v$ for an arbitrary vector $v \in \mathbb{R}^d$ with $O(nd)$ time.

Because the hession matrix is in the form like $\sum a x_i x_i^T + I$,we can calculate the $\nabla^2 f(w)v$ by

$$(\sum a x_i x_i^T + I)v = \sum a x_i x_i^T v + v = \sum a x_i (x_i^T v) + v$$

this will avoid doing the d by d matrix calculation.

(c) Use the martix-vector product derived in (c) to implement Conjugate Gradient method (See Algorithm 1) for solving the linear system $(\nabla^2 f(w_0)^{-1}\nabla f(w_0))$ up to $||r||/||r_o|| \leq 10^{-3}$ (where r is the residual in the algorithm). Report how much time does the algorithm take on the breast-cancer dataset. Report $||\triangle x^{CG}||_2$ where $\triangle x^{CG}$ is the output of CG.

```
x=read.table('C:/Users/Chan/Desktop/Files/STA250/Homework 2/hw2_data/X_cancer')
y=read.table('C:/Users/Chan/Desktop/Files/STA250/Homework 2/hw2_data/y_cancer')
n=nrow(x)
d=ncol(x)
X=cbind(y,x)
```

Algorithm 1

```
CG=function(w){
  b=-fw1(w)#this is the b in the Algorithm

  output=integer(d)
  r=b-0#Ax=0
  r0=sqrt(sum(r^2))
  p=r

  while(sqrt(sum(r^2))/r0>10^(-3)){

    #use the product derived in (b)
    fw=rowSums(apply(X,1,function(X){
      e=exp(-X[1]*X[-1]%*%w)
      t(X[-1])*as.numeric(X[-1]%*%p)*as.numeric((X[1]^2*e)/(1+e)^2)
    }))
    Ap=fw+p#Ap_k in the Algorithm

    a=as.numeric(t(r)%*%r/(t(p)%*%Ap))
    output=output+a*p
    r2=r-a*Ap
    b=as.numeric(t(r2)%*%r2/t(r)%*%r)
    p=r2+b*p
    r=r2
  }
  output
}
```

Report how much time does the algorithm take on the dataset.

```
system.time(CG(numeric(d)))
```

```
##    user  system elapsed
##    0.97    0.00    0.97
```

the iteration times

```
CG2=function(w){
  b=fw1(w)#this is the b in the Algorithm

  output=integer(d)
  r=b-0#Ax=0
  r0=sqrt(sum(r^2))
  p=r
  i=0#count for iteration
```

```
  while(sqrt(sum(r^2))/r0>10^(-3)){
    i=i+1
    #use the product derived in (b)
    fw=rowSums(apply(X,1,function(X){
      e=exp(-X[1]*X[-1]%*%w)
      t(X[-1])*as.numeric(X[-1]%*%p)*as.numeric((X[1]^2*e)/(1+e)^2)
    }))
    Ap=fw+p#Ap_k in the Algorithm

    a=as.numeric(t(r)%*%r/(t(p)%*%Ap))
    output=output+a*p
    r2=r-a*Ap
    b=as.numeric(t(r2)%*%r2/t(r)%*%r)
    p=r2+b*p
    r=r2
  }
  i#change the output to the iteration times
}
CG2(numeric(d))
```

```
## [1] 12
```

Report $||\triangle x^{CG}||_2$ where $\triangle x^{CG}$ is the output of CG.

```
norm(CG(numeric(d)),"2")
```

```
## [1] 0.177959
```

(d)Implement the Newton-CG algorithm (See Algorithm 2). Initial from $w^0 = 0$ and run 20 iterations. What is the objective function value $f(w^20)$? Plot the iteration vs error figure: x-axis is k=1,. . . ,18 abd y-axis is corresponding error $\log(f(w^k) - f(w^*))$. In practice it is impossible to get the exact optimal solution w^*, so instead we assume $f(w^*) = f(w^{20})$ when we plot the figure.

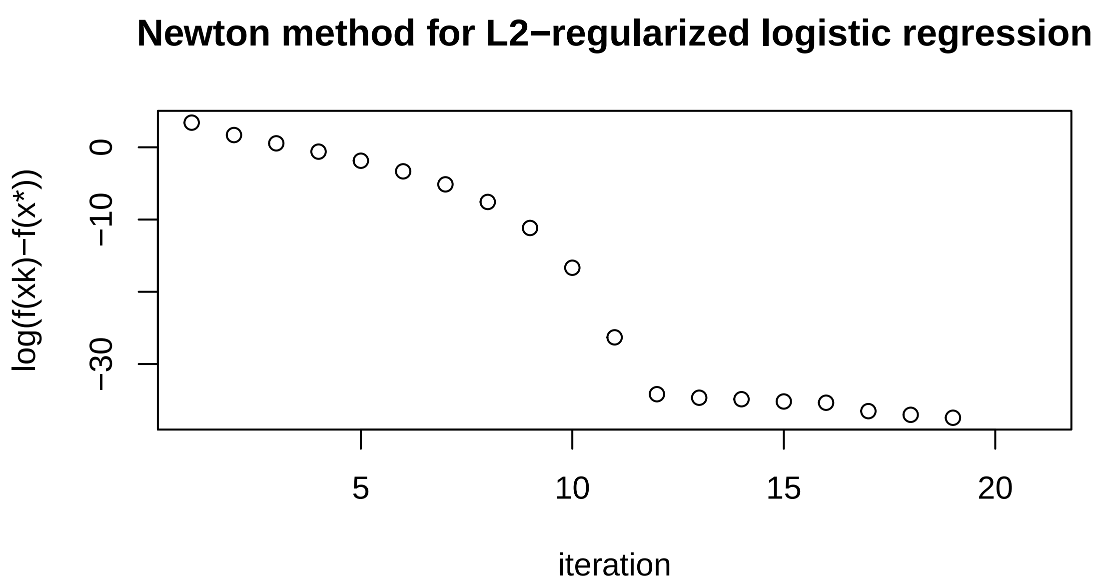
```
#set the iteration times
k=20
#initial w
w=matrix(0, d, 1)
#To record w
wlist=matrix(0,d,(k+1))
wlist[,1]=w

for(j in 1:k){
  D=CG(w)
  alpha=1
  while(fw(w+alpha*D)>fw(w)+0.01*alpha*t(D)%*%fw1(w)){
    alpha=alpha/2
  }
  w=w+alpha*D#the new w
  wlist[,j+1]=w#record w
}
```

```
flist=vector(mode='numeric',k+1)
for (j in 1:(k+1)) {
  flist[j]=fw(wlist[,j])}
```

```
min(flist)
```

```
## [1] 0.2158171
```

```r
plot(log(flist-min(flist)),xlab = "iteration",ylab = "log(f(xk)-f(x*))",main = "Newton method for L2-re
```



Problem 2. Nonnegative Matrix Factorization

Given an input matrix $X \in \mathbb{R}^{m\times n}$,we try to factorize it into $X \approx WH^T$ by solving

$$(NMF) \min_{W\in\mathbb{R}^{m\times k},H\in\mathbb{R}^{n\times k}} \frac{1}{2}||A - WH^T||_F^2 + \frac{\lambda}{2}||W||_F^2 + \frac{\lambda}{2}||H||_F^2$$

$$s.t.W_{ij} \geq 0, H_{ij} \geq 0\forall i,j$$

We will use the cbcl dataset in "cbcl.txt". In this data, m=361,n=2,429, and we set k=49.

(a) Apply a block-coordinate descent algorithm to solve NMF. At each iteration, we first fix W and update H, and then fix H and update W. For each subproblem, update W (or H) using projected gradient descent with 3 steps (see Algorithm 3). USe random initialization for W,H. Run the algorithm for 50 iterations and plot the objective value vs time cureve. Report the final objective function value you get.

First we need to derive the gradient

$$\nabla_W f(W,H) = (WH^T - A)H + \lambda W$$

$$\nabla_H f(W,H) = (WH^T - A)^T W + \lambda H$$

```r
fW=function(W,H){
  (W%*%t(H)-A)%*%H+lambda*W
}

fH=function(W,H){
  t((W%*%t(H)-A))%*%W+lambda*H
}
```

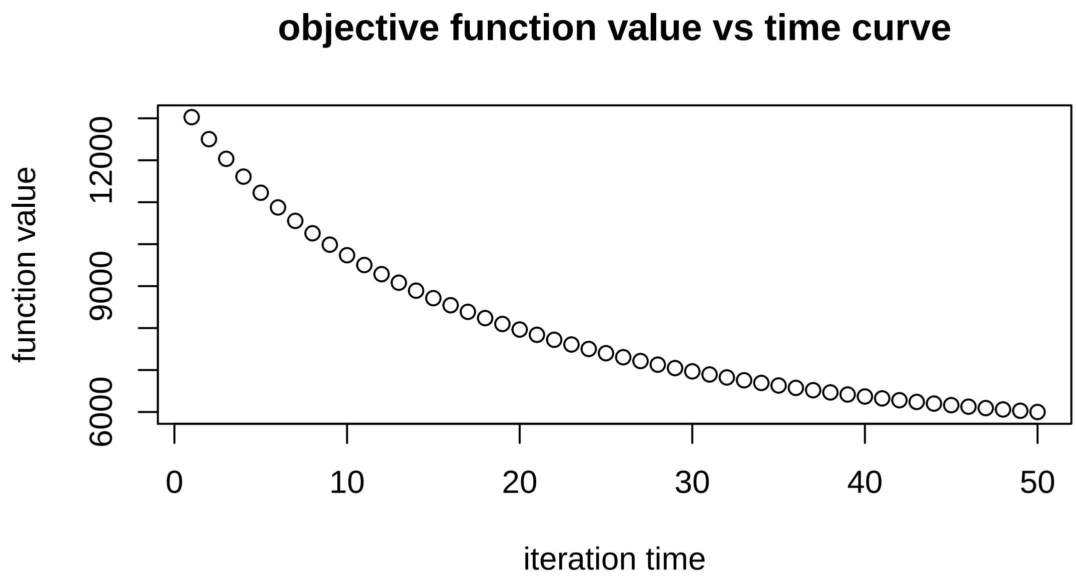
```r
#objective function
f=function(W,H){
  norm(as.matrix(A-W%*%t(H)),"F")^2/2+lambda/2*norm(W,"F")^2+lambda/2*norm(H,"F")^2
}
```

```r
A=read.table('C:/Users/Chan/Desktop/Files/STA250/Homework 2/hw2_data/cbcl.txt')
A=as.matrix(A)
m=nrow(A)
n=ncol(A)
k=49
```

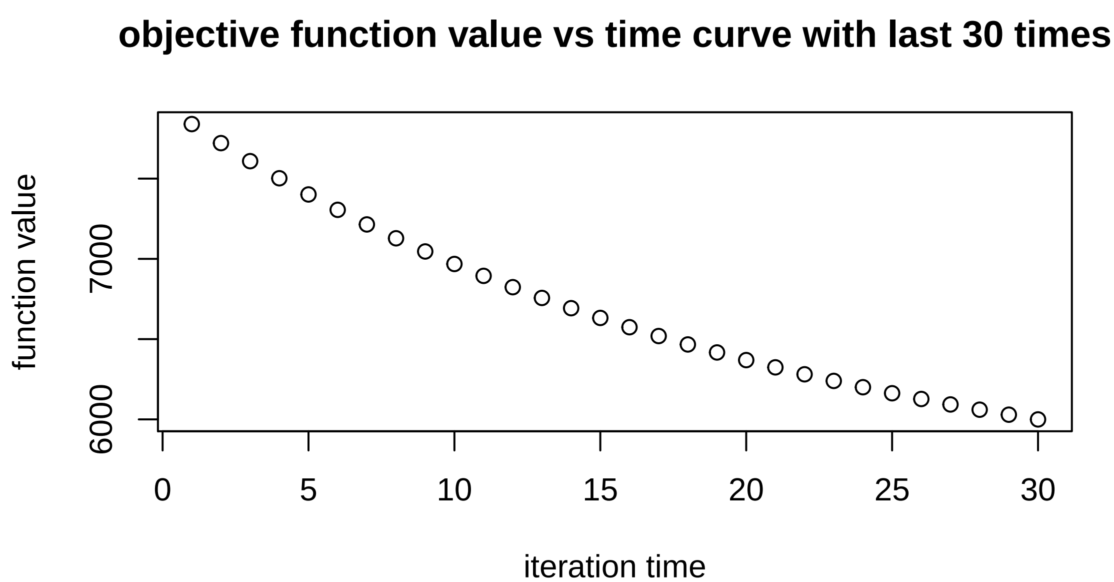
```r
#initial W and H randomly
set.seed(100)
W=matrix(runif(m*k,0,.1),m, k)
H=matrix(runif(n*k,0,.1),n, k)

#set eta
eta=0.0001
#set lambda
lambda=1
flist=f(W,H)

for(i in 1:50){
  for(j in 1:3){
    W=W-eta*fW(W,H)#the new w
    W[W<0]=0
    }
  for(j in 1:3){
    H=H-eta*fH(W,H)#the new w
    H[H<0]=0
  }
  flist=cbind(flist,f(W,H))
}
```

function value vs time curve and the final objective function value

```r
plot(as.vector(flist[-1]),xlab = "iteration time",ylab = "function value",
     main = "objective function value vs time curve")
```




```
plot(as.vector(flist[-(1:21)]),xlab = "iteration time",ylab = "function value",
     main = "objective function value vs time curve with last 30 times")
```




```
f(W,H)
```

```
## [1] 5999.648
```

(b) Now try block coordinate descent by defining each column of W,H as a block. So there are totally 2k blocks (See Algorithm 4). For updating w_i(for some $1 \leq i \leq k$), we want to minimize the function with respect to w_i. Derive the corresponding subproblem and show the close form solution of w_i. Similarly run 50 iterations, report the final objective function value, and plot the objective function value vs time curve. Compare the two methods and discuss your findings.

Fix H and the remain column of W, the subproblem is that

$$\min_{w_i} \frac{1}{2}||A^* - w_i h_i^T||_F^2 + \frac{\lambda}{2}||w_i||_F^2$$

where

$$A^* = A - \sum_{k \neq i} w_k h_k$$

First we need to derive the gradient

$$\nabla_{W_i} f(W, H) = (w_i h_i^T - A^*) h_i + \lambda w_i$$

set the gradient to zero, and we get

$$w_i = \frac{A^* h_i}{h_i^T h_i + \lambda}$$

we also need to projection to let $w_i \geq 0$

$$\nabla_{h_i} f(W, H) = (WH^T - A)^T w_i + \lambda h_i$$

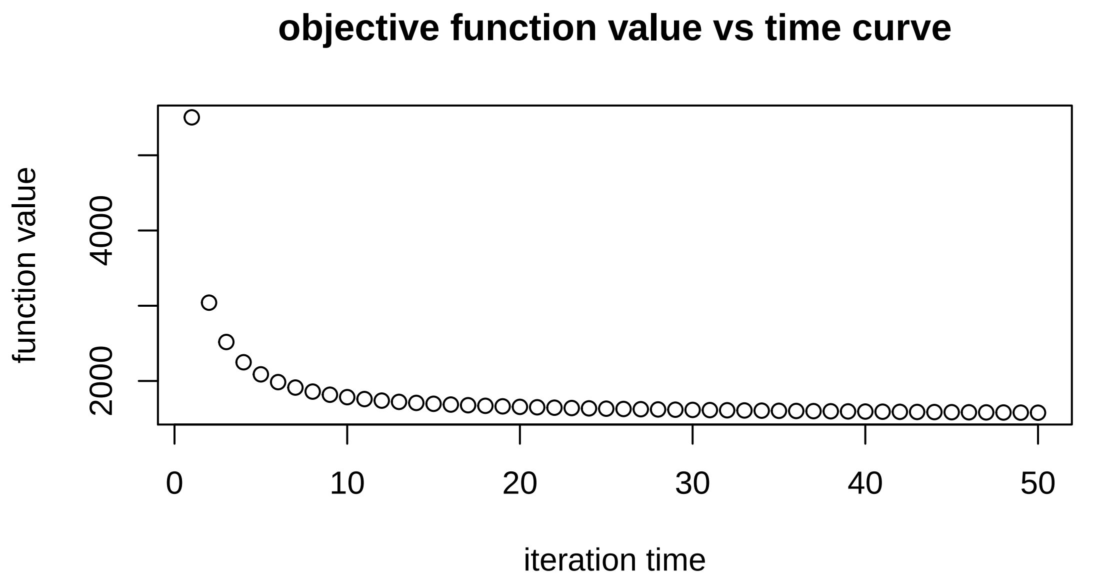
similarly

$$h_i = \frac{A^{*T} w_i}{w_i^T w_i + \lambda}$$

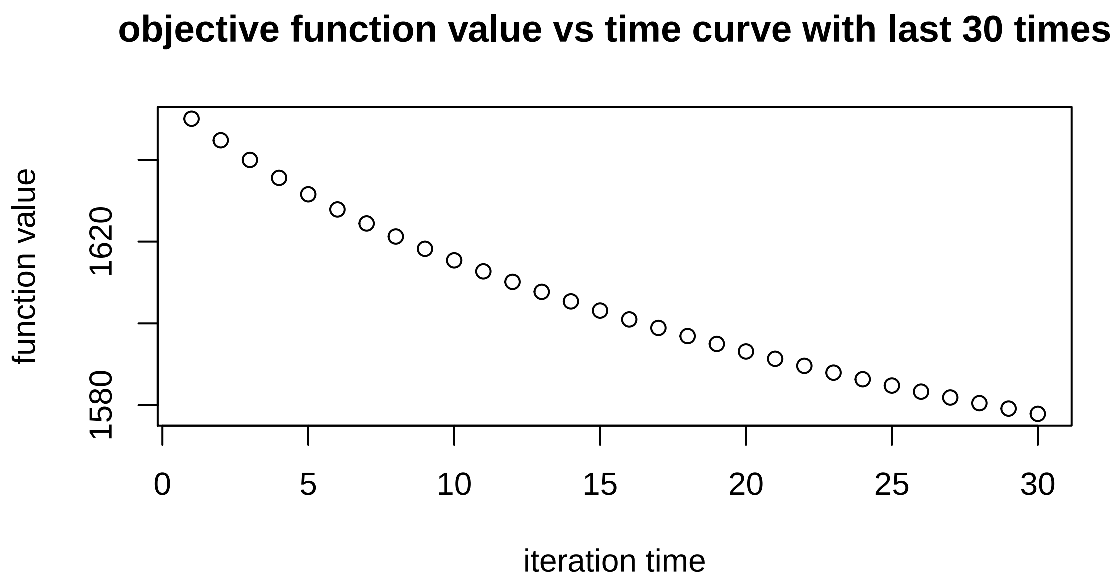
```
#initial W and H randomly,seed is set so they are the same as (a)
set.seed(100)
W=matrix(runif(m*k,0,.1), m, k)
H=matrix(runif(n*k,0,.1), n, k)

#set eta
eta=0.0001
#set lambda
lambda=1
flist=f(W,H)

for(i in 1:50){
  for(j in 1:k){
    A2=A-W%*%t(H)+W[,j]%*%t(H[,j])
    W[,j]=A2%*%H[,j]/as.numeric(t(H[,j])%*%H[,j]+lambda)#the new wj
    W[W<0]=0
    }
  for(j in 1:k){
    A2=A-W%*%t(H)+W[,j]%*%t(H[,j])
    H[,j]=t(A2)%*%W[,j]/as.numeric(t(W[,j])%*%W[,j]+lambda)#the new w
    H[H<0]=0
  }
  flist=cbind(flist,f(W,H))
}
```

function value vs time curve and the final objective function value

```
plot(as.vector(flist[-1]),xlab = "iteration time",ylab = "function value",
     main = "objective function value vs time curve")
```




```r
plot(as.vector(flist[-(1:21)]),xlab = "iteration time",ylab = "function value",
     main = "objective function value vs time curve with last 30 times")
```




```r
f(W,H)
```

```
## [1] 1577.892
```