HW 4
CSE 250A
Weiwei Li
A53107958

1. a) $L = \sum_{t=1}^{T} \log P(y_t | \vec{x}_t)$

$= \sum_{t=1}^{T} \log \left(P(y_t=1 | \vec{x}_t)^{y_t} \cdot P(y_t=0 | \vec{x}_t)^{1-y_t} \right)$

$= \sum_{t=1}^{T} \left[y_t \log(P(y_t=1 | \vec{x}_t)) + (1-y_t) \log(P(y_t=0 | \vec{x}_t)) \right]$

$= \sum_{t=1}^{T} \left[y_t \log g(\vec{w}\vec{x}_t) + (1-y_t) \log(1 - g(\vec{w}\vec{x}_t)) \right]$

$\frac{\partial L}{\partial w_i} = \sum_{t=1}^{T} \left[\frac{y_t}{g(\vec{w}\vec{x}_t)} \cdot g'(\vec{w}\vec{x}_t) \cdot x_{it} + \frac{1-y_t}{1-g(\vec{w}\vec{x}_t)} \cdot (-g'(\vec{w}\vec{x}_t)) \cdot x_{it} \right]$

$= \sum_{t=1}^{T} \left[\frac{y_t}{P_t} g'(\vec{w}\vec{x}_t) \cdot x_{it} + \frac{1-y_t}{1-P_t} (-g'(\vec{w}\vec{x}_t)) x_{it} \right]$

$= \sum_{t=1}^{T} \left[\frac{g'(\vec{w}\vec{x}_t)}{P_t(1-P_t)} \right] (y_t - P_t) x_{it}$

b) $g(z) = \frac{1}{1+e^{-z}}$, then $g'(z) = g(z)g(-z)$

$\frac{\partial L}{\partial w_i} = \sum_{t=1}^{T} \frac{g(\vec{w}\vec{x}_t) \cdot (1 - g(\vec{w}\vec{x}_t))}{g(\vec{w}\vec{x}_t)(1-g(\vec{w}\vec{x}_t))} (y_t - g(\vec{w}\vec{x}_t)) x_{it}$

$= \sum_{t=1}^{T} (y_t - g(\vec{w}\vec{x}_t)) x_{it}$

So the result in a) reduces to the result in lecture when considering about sigmoid function.

2 a). $L = \sum_t \log P(y_t | \vec{x}_t) = \sum_t \log \prod_{i=1}^{C} P_{it}^{y_{it}} = \sum_t \sum_{i=1}^{C} y_{it} \log P_{it}$

ith lablel: $\frac{\partial L}{\partial w_{\alpha i}} = \sum_t y_{it} \frac{1}{P_{it}} P'_{it}$

$= \sum_t y_{it} \frac{1}{P_{it}} \frac{e^{\vec{w}_i \vec{x}_t} (1 - e^{\vec{w}_i \vec{x}_t}) x_{\alpha t}}{(\sum_j e^{\vec{w}_j \vec{x}_t})^2}$

$= \sum_t y_{it} (1 - P_{it}) x_{\alpha t}$

kth label exrept ith: $\frac{\partial L}{\partial w_{\alpha i}} = \sum_t y_{kt} \cdot \frac{1}{P_{kt}} \cdot P'_{kt}$

$= \sum_t y_{kt} \cdot \frac{1}{P_{kt}} \left(- \frac{e^{\vec{w}_k \vec{x}_t} e^{\vec{w}_i \vec{x}_t}}{(\sum_j e^{\vec{w}_j \vec{x}_t})^2} \right) x_{\alpha t}$

$= \sum_t - y_{kt} P_{it} x_{\alpha t}$

$$\frac{\partial L}{\partial W_{\alpha i}} = \frac{\partial L}{\partial W_{\alpha i}} + \sum_{k \neq i} \frac{\partial L}{\partial W_{\alpha i}}$$

(ith label; kth label except ith)

$$= \sum_t y_{it}(1-P_{it}) x_{\alpha t} + \sum_t \sum_{k \neq i} -y_{kt} P_{it} x_{\alpha t}$$

$$= \sum_t \left(y_{it} x_{\alpha t} - \sum_{k=1}^{C} y_{kt} P_{it} x_{\alpha t} \right)$$

$$= \sum_t (y_{it} x_{\alpha t} - P_{it} x_{\alpha t})$$

$$= \sum_t (y_{it} - P_{it}) x_{\alpha t}$$

$$\therefore \frac{\partial L}{\partial W_i} = \sum_t (y_{it} - P_{it}) \vec{x}_t$$

3. a) $f'(x_n) = \alpha(x_n - x_*) = \alpha \varepsilon_n$

$$\varepsilon_n = x_n - x_*$$

$$= x_{n-1} - \eta f'(x_{n-1}) - x_*$$

$$= x_{n-1} - \eta \alpha \varepsilon_{n-1} - x_*$$

$$= \varepsilon_{n-1} + x_* - \eta \alpha \varepsilon_{n-1} - x_*$$

$$= (1 - \eta\alpha) \varepsilon_{n-1}$$

$$\therefore \varepsilon_n = (1-\eta\alpha)^n \varepsilon_0$$

Q. b) by a). if we want the update rule converge to the minimum at x^*. we need

$$-1 < 1 - \eta\alpha < 1 \Rightarrow \quad 0 < \eta < \frac{2}{\alpha}$$

If we want the fastest convergence. then we need $1 - \eta\alpha = 0 \Rightarrow \eta = \frac{1}{\alpha}$.

$$\eta = \frac{1}{\alpha} = \frac{1}{f''(x_n)}$$

c)

$$\varepsilon_{n+1} = x_{n+1} - x_*$$

$$= x_n - \eta\alpha\varepsilon_n + \beta(x_n - x_{n-1}) - x_*$$

$$= \varepsilon_n + x_* - \eta\alpha\varepsilon_n + \beta(\varepsilon_n + x_* - \varepsilon_{n-1} - x_*) - x_*$$

$$= (1 + \beta - \eta\alpha)\varepsilon_n - \beta\varepsilon_{n-1}$$

d) $\varepsilon_{n+1} = (1 - 1\cdot\frac{4}{9} + \frac{1}{9})\varepsilon_n - \frac{1}{9}\varepsilon_{n-1} = \frac{2}{3}\varepsilon_n - \frac{1}{9}\varepsilon_{n-1}$

$\lambda^{n+1}\varepsilon_0 = \frac{2}{3}\lambda^n\varepsilon_0 - \frac{1}{9}\lambda^{n-1}\varepsilon_0$

$\lambda^{n+1} - \frac{2}{3}\lambda^n + \frac{1}{9}\lambda^{n-1} = 0$

when $n=1$: $\lambda^2 - \frac{2}{3}\lambda + \frac{1}{9} = 0 \Rightarrow \lambda = \frac{1}{3}$

If $\beta = 0$: $\varepsilon_{n+1} = (1 - \frac{4}{9})\varepsilon_n = \frac{5}{9}\varepsilon_n$.

$\lambda^{n+1}\varepsilon_0 = \frac{5}{9}\lambda^n\varepsilon_0$

$\lambda^{n+1} - \frac{5}{9}\lambda^n = 0$

$\lambda = \frac{5}{9}$

$\beta = 1$: $\varepsilon_n = (\frac{1}{3})^n\varepsilon_0$.

$\beta = 0$: $\varepsilon_n = (\frac{5}{9})^n\varepsilon_0$.

$0 < (\frac{1}{3})^n < (\frac{5}{9})^n < 1$ when $n \geq 1$

So the gradient descent learning rule with momentum test converges more fast.

4) a) $g'(x_n) = 2k(x_n - x_*)^{2k-1}$

$g''(x_n) = (2k)(2k-1)(x_n - x_*)^{2k-2}$

$x_{n+1} = x_n - \frac{x_n - x_*}{2k-1} = \frac{2k-2}{2k-1}x_n + \frac{1}{2k-1}x_*$

$\varepsilon_n = |x_n - x_*| = |\frac{2k-2}{2k-1}x_{n+1} + \frac{1}{2k-1}x_* - x_*|$

$= |\frac{2k-2}{2k-1}x_{n+1} + \frac{2-2k}{2k-1}x_*|$

$= \frac{2k-2}{2k-1}|x_{n+1} - x_*|$ $\because k > 0 \quad k \in \mathbb{Z}$.

$= \frac{2k-2}{2k-1}\varepsilon_{n+1}$

$\therefore \varepsilon_n = (\frac{2k-2}{2k-1})^n\varepsilon_0$

b) $\varepsilon_n = \left(\frac{2k-2}{2k-1}\right)^n \varepsilon_0 \le \delta \varepsilon_0$

$n \log \frac{2k-2}{2k-1} \le \log \delta$

$n \ge \log \delta / \log\left(\frac{2k-2}{2k-1}\right)$

$n \ge \frac{\log \delta}{-\frac{1}{2k-1}}$ since $\log \frac{2k-2}{2k-1} \le \frac{2k-2}{2k-1} - 1 = -\frac{1}{2k-1}$

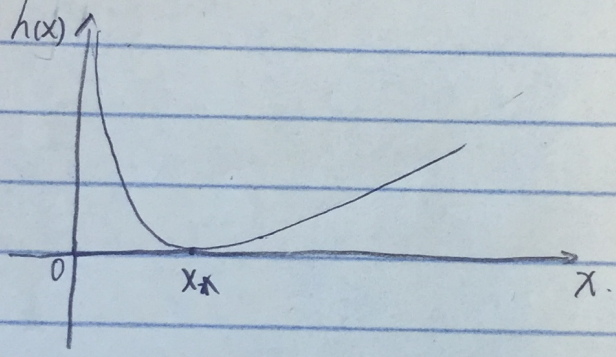
$n \ge (2k-1)\log\left(\frac{1}{\delta}\right)$

c) $h(x) = x_* \log\left(\frac{x_*}{x}\right) - x_* + x \qquad x_* > 0$

$h'(x) = x_* \cdot \frac{x}{x_*} \cdot -x_* \cdot \frac{1}{x^2} + 1 = 1 - \frac{x_*}{x} = 0 \Rightarrow x = x_*$

$h''(x) = x_* \cdot \frac{1}{x^2} > 0$ since $x_* > 0$

So $x = x_*$ is the minimum point.



$-x_* < x - x_* < x_* \Rightarrow 0 < x < 2x_*$

d) $\rho_n = \frac{x_n - x_*}{x_*}$ $\qquad x_{n+1} = x_n - \frac{(x_n - x_*)/x_n}{x_*/x_n^2} = x_n - \frac{x_n^2 - x_n x_*}{x_*} = 2x_n - \frac{x_n^2}{x_*}$

$= \frac{2x_{n-1} - \frac{x_{n-1}^2}{x_*} - x_*}{x_*}$

$= -\frac{x_{n-1}^2}{x_*^2} + \frac{2x_{n-1}}{x_*} - 1$

$= -\left(\frac{x_{n-1}}{x_*} - 1\right)^2$

$= -\left(\frac{x_{n-1} - x_*}{x_*}\right)^2$

$= -\rho_{n-1}^2$ $\qquad \therefore \rho_n = -\rho_0^{2^n}$

If we want newton's method converge, then we need $-1 < \rho_0 < 1 \Leftrightarrow$

$-1 < \frac{x_0 - x_*}{x_*} < 1 \Leftrightarrow \quad 0 < x_0 < 2x_*$

250hw2- 4 & 5

Weiwei Li(A53107958)

1 4.5 Stock market prediction

1.1 (a) Linear coefficients

The MLE of a1 is 0.9452001, a2 is 0.0197424, a3 is -0.013645, a4 is 0.0467813.

1.2 b) Mean squared prediction error

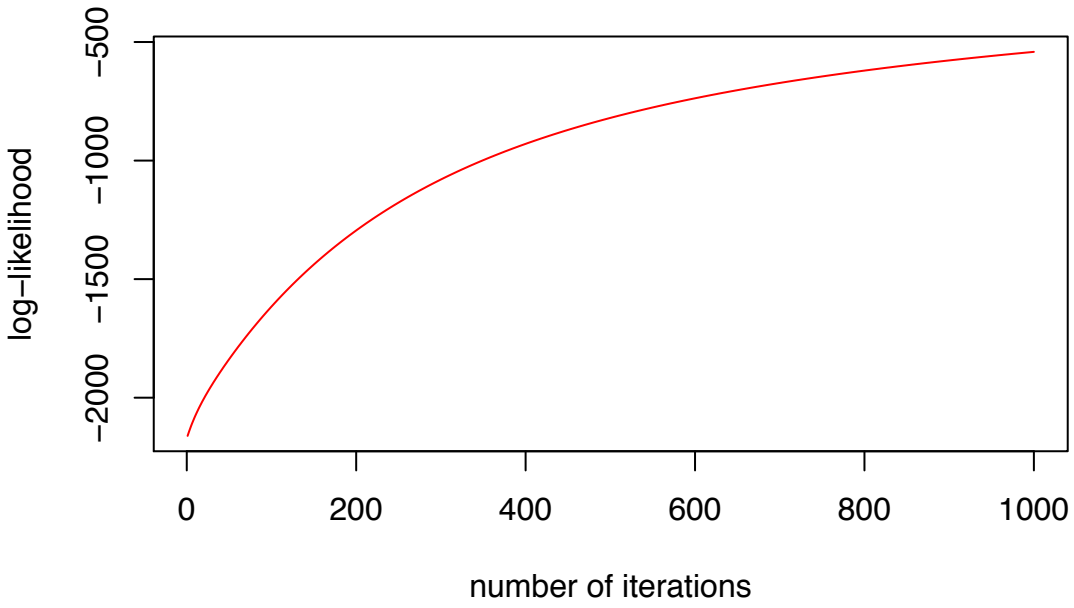
The mean squared error of 2000 is 13918.63.

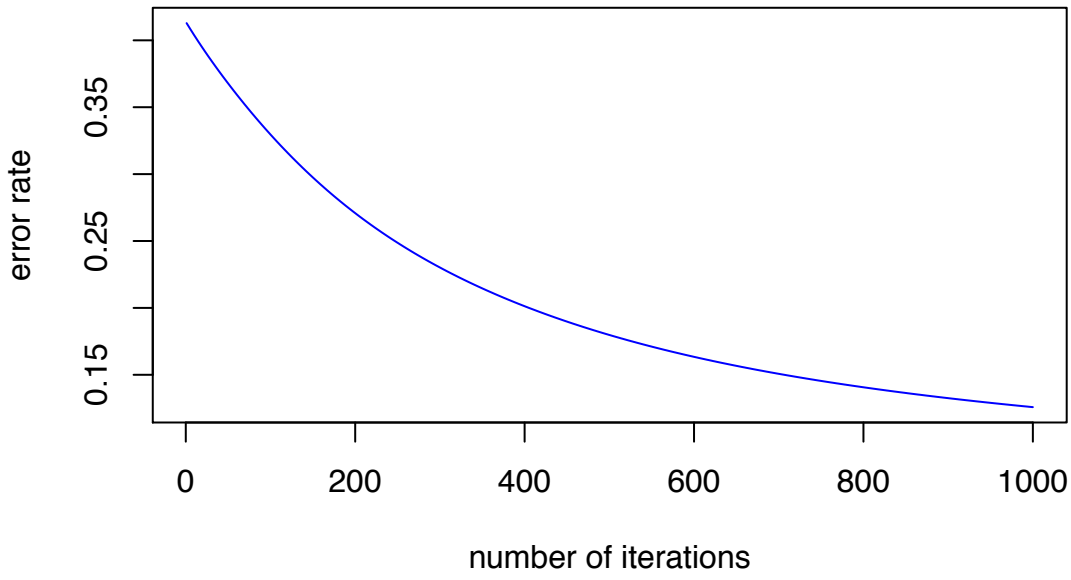
The mean squared error of 2001 is 3018.2678422.

I would not recommend this linear model for stock market prediction, because the error is big.

2 4.6 Handwritten digit classification

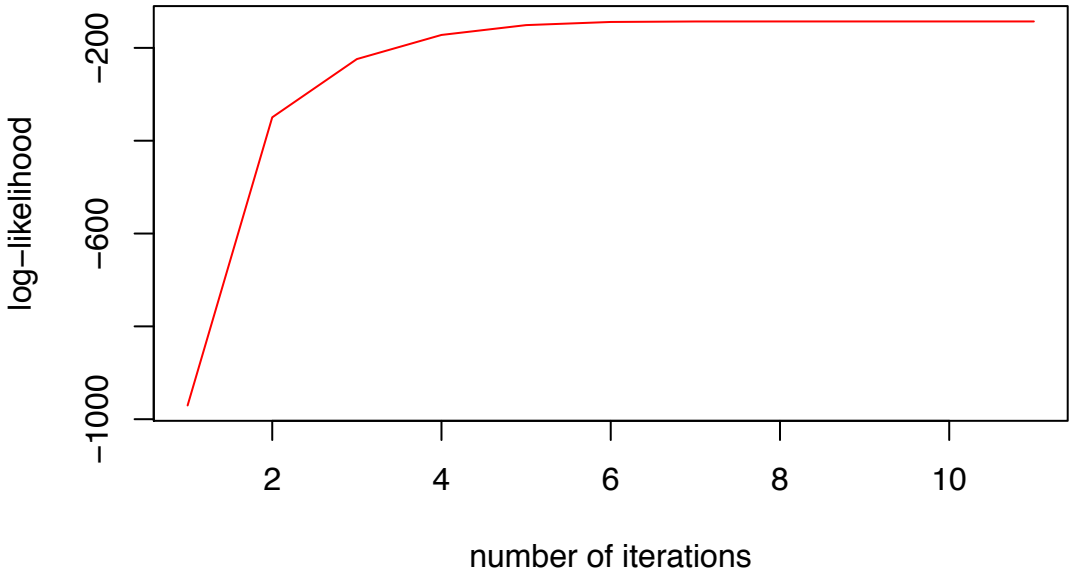
2.1 a) Gradient ascent

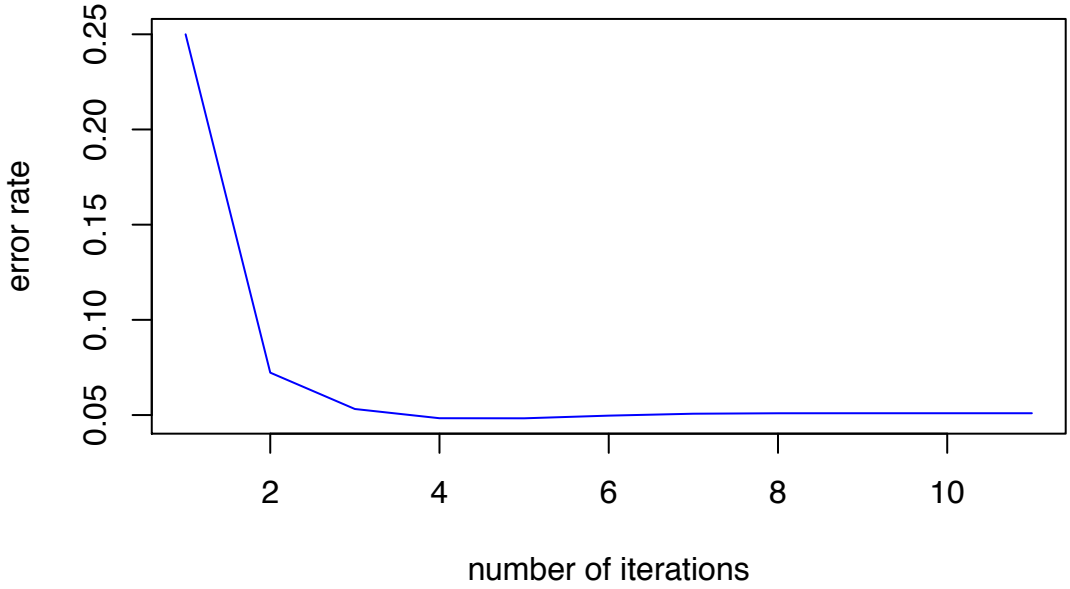





```
           [,1]       [,2]       [,3]       [,4]        [,5]       [,6]
[1,] -1.3837103  0.2951865 -1.4146963 -0.5218373 -1.20880135  0.7732218
[2,]  1.7963441 -1.9147015  1.5059125  0.2297306  1.09813399 -0.8982577
[3,]  0.3842503  0.6573507  1.2150253  0.2072060  0.53562075 -2.1320498
[4,]  0.1828994  1.1343126 -0.6583342  0.4899277 -0.16069051 -0.3555325
[5,]  0.5688929  0.5434954  0.6308846 -0.3446184 -0.94077157  0.7923339
[6,]  1.2588283  0.5128150 -0.4562587  2.2280189  0.09122572  1.1919431
[7,]  2.0174025  0.5702295  0.4400113 -1.8818116  0.17673742 -0.1938166
[8,]  0.5537133 -0.9679566 -1.3566584  0.9798271 -0.34945396  0.3380507
           [,7]       [,8]
[1,] -1.7609249  2.6655138
[2,]  0.3805005 -0.4228724
[3,] -1.3413106 -0.6634776
[4,] -1.4383112  1.1680715
[5,] -0.2053746  0.1315388
[6,]  0.5729720  1.2734487
[7,] -2.1368357  0.9807177
[8,] -0.4403554  0.6848544
```

2.2 Newton:






```
           [,1]        [,2]       [,3]        [,4]        [,5]        [,6]
[1,] -2.0734746 -1.11965927 -1.9514166 -1.62430090 -1.84023358 -0.18588219
[2,]  0.9388545 -0.81552578  1.2222572 -0.15407065 -0.30956348  0.69589794
[3,]  2.8857775  1.48857693  0.9496304  0.05942676  0.98704155 -2.44728470
[4,]  1.3765895  1.49832356  1.1732132 -0.37058086 -0.89774640 -2.19800332
[5,]  1.0066704  0.59183628 -0.7502970 -0.32948543 -0.36713085  0.28090254
[6,]  2.0229304 -1.17400459 -0.1322109  1.85987315 -0.12253932 -0.76555564
[7,]  0.7963150 -0.07837087  0.5771092  0.46984959  0.04860286  0.04161249
[8,]  0.9257582  0.68001321 -1.7880423  4.24524953  0.07141610  1.17593224
           [,7]        [,8]
[1,]  0.98184048  2.61362832
[2,] -1.52354073 -0.37616390
[3,] -3.83888486 -3.53932865
```

```
[4,]  0.84874748 -0.03796541
[5,] -0.20682627  0.83167733
[6,]  0.34344151 -1.87539663
[7,] -0.28925360 -1.62666650
[8,] -0.06759093  0.88356380
```

The error rate for the test files with gradient ascent is 0.1258213.

The error rate for the test files with Newtons method is 0.0509484.

I use both of gradient ascent and Newtons method. plot out the log-likelihood on several iterations of the algorithm, as well as the percent error rate on the images in these files.

And I found that although the diference between the error rates from the two methods are small, the convergence rate for Newton's method is much faster. Newtons method converges at about 10 iterations, while gradient ascent converges at about 1000 iterations. Also the weight vector for the two methods are shown in the file.

```
#4.5 Stock market prediction
#a)
Data00 = read.table("/Users/weiweili/Dropbox/CSE 250A/hw4 data/nasdaq00.txt")
Data01 = read.table("/Users/weiweili/Dropbox/CSE 250A/hw4 data/nasdaq01.txt")

N00 = dim(Data00)[1]
N01 = dim(Data01)[1]

library(stats4)

logP = matrix(nrow=N00-4, ncol=1)
L = function(a1, a2, a3, a4){
 for ( i in 5 : N00){
  logP[i-4] = log(1/(sqrt(2*pi))) + (-1/2)*(Data00[i,1]-a1*Data00[i-1,1]-a2*Data00[i-2,1]-
a3*Data00[i-3,1]-a4*Data00[i-4,1])^2
 }
  return(-sum(logP))
}

MLE = mle(L,  start = list(a1=0,a2=0,a3=0, a4=0))
a1 = as.numeric(MLE@coef[1])
a2 = as.numeric(MLE@coef[2])
a3 = as.numeric(MLE@coef[3])
a4 = as.numeric(MLE@coef[4])

#a1=0.94520006    a2=0.01974237     a3= -0.01364498     a4=0.04678134

#b)

SE= matrix(nrow=N00, ncol=2)
D = matrix(nrow=N00, ncol=2)
D[1:4,1]=Data00[1:4,1]
D[1:4,2]=Data01[1:4,1]
SE[1:4,1] = (Data00[1:4,1]-D[1:4,1])^2
SE[1:4,2] = (Data01[1:4,1]-D[1:4,2])^2
for (i in 5: N00){
  D[i,1] = Data00[i,1]-a1*Data00[i-1,1]-a2*Data00[i-2,1]-a3*Data00[i-3,1]-a4*Data00[i-4,1]
  SE[i,1] = D[i,1]^2
  D[i,2] = Data01[i,1]-a1*Data01[i-1,1]-a2*Data01[i-2,1]-a3*Data01[i-3,1]-a4*Data01[i-4,1]
  SE[i,2] = D[i,2]^2
}
MSE00 = sum(SE[,1])/(N00-4)
MSE01 = sum(SE[,2][-N00])/(N01-4)

# MSE00 = 13918.63        MSE01= 3018.268
# I would not recommend this linear model for stock market prediction, because the error is big.
```

```
# Handwritten digit classification
#a)
trn3 = read.table("~/Dropbox/Vivi/vivi homework/CSE 250A/hw4 data/newTrain3.txt")
tst3 = read.table("~/Dropbox/Vivi/vivi homework/CSE 250A/hw4 data/newTest3.txt")
trn5 = read.table("~/Dropbox/Vivi/vivi homework/CSE 250A/hw4 data/newTrain5.txt")
tst5 = read.table("~/Dropbox/Vivi/vivi homework/CSE 250A/hw4 data/newTest5.txt")


test = rbind(as.matrix(tst3), as.matrix(tst5))

train = rbind(as.matrix(trn3), as.matrix(trn5))

#log-likelihood
L = function(w,x,y){
  # x predictor matrix: m*n
  # y binary: m*1 vector
  # w weight: n*1 vector
  # L first colum is loglikelihood, the rest columns are gradients
  q = 1/(1+exp( -x%*%w ))   # dim m*1
  l= y*log(q)+(1-y)*log(1-q) #dim m*1
  return(sum(l))
}

#gradient
g = function(w,x,y){
  # x predictor matrix: m*n
  # y binary: m*1 vector
  # w weight: n*1 vector
  # g gradients: n*1
  q = 1/(1+exp( -x%*%w ))   # dim m*1
  g= diag(as.vector(y-q))%*%x  #dim m*n
  return( colSums(g) )
}


#hessian
hessian = function(w,x){
  # w weight: n*1 vector
  # x predictor matrix: m*n
  q = 1/( 1+exp(-x%*%w) )   # dim m*1
  c = q*(1-q) # dim m*1
  n = ncol(x)
  H =  matrix(NA,n,n)
  for (i in 1:n){
    for (j in 1:i){
      H[i,j] = -sum(c*x[,i]*x[,j])
      H[j,i] = H[i,j]
    }
  }
  return(H)
}


error = function(w,x,y) {
  # x predictor matrix: m*n
  # y binary: m*1 vector
  # w weight: n*1 vector
  # L first colum is loglikelihood, the rest columns are gradients
  q = 1/(1+exp( -x%*%w ))   # dim m*1
  #l = abs(y-q)   #error
  l=  (y - q)^2 #dim m*1  #error
```

```
  return(sum(l)/nrow(x))
}




y1 = matrix(0,nrow(trn3), 1)
y2 = matrix(1,nrow(trn5), 1)
y =  rbind(y1, y2)
test.y = matrix(0,nrow(tst3),1)
test.y = rbind(test.y, matrix(1,nrow(tst5), 1))



#gradient ascent:
w=matrix(rnorm(64,0,1),nrow=64, ncol=1)
k = 0.02/nrow(train)

lh2 = L(w, train, y)
dif = 1
N = 100
LH = matrix(nrow = N, ncol = 1)
err = matrix(nrow = N, ncol = 1)
i = 1
while(dif >= 0 & i<=N){
  lh = lh2
  LH[i] = lh
  err[i] = error(w,test,test.y)
  oldw = w
  grad = g(w, train, y)
  w = w+k*(grad)
  lh2 = L(w, train, y)
  dif = lh2 - lh
  i=i+1
}

if(dif < 0){w = oldw}
plot(seq(1,i-1),LH[1:i-1], type = "l", col = "red", xlab = "number of iterations", ylab = "log-
 likelihood")
plot(seq(1,i-1),err[1:i-1], type = "l", col = "blue", xlab = "number of iterations", ylab = "error
 rate")




#Newton:
norm_vec <- function(x) sqrt(sum(x^2))

w=matrix(0,nrow=64, ncol=1)

lh2 = L(w, train, y)
dif = 1
N = 40
LH = matrix(nrow = N, ncol = 1)
err = matrix(nrow = N, ncol = 1)
i = 1
while( dif > 0 & i<=N){
  lh = lh2
```

```
  LH[i] = lh
  err[i] = error(w,test,test.y)
  oldw = w
  grad = g(w, train, y)
  H = hessian(w,train)
  if ( det(H) < 1e-6 ) {
    aH = H + 0.001*diag(1,ncol(train),ncol(train))
    k2 = solve(aH,grad)
  } else {
    k2 = solve(H,grad)
  }
  w = w-k2
  lh2 = L(w, train, y)
  #grad2 = g(w,train,y)
  #dif = norm_vec(grad2 - grad)
  dif = lh2 - lh
  i=i+1
}


if(dif < 0){w = oldw}
plot(seq(1,i-1),LH[1:i-1], type = "l", col = "red", xlab = "number of iterations", ylab = "log-
 likelihood")
plot(seq(1,i-1),err[1:i-1], type = "l", col = "blue", xlab = "number of iterations", ylab = "error
 rate")
```