**SGD for Logistic Regression**

**(A)** Proof:

$\sum_{i=1}^{n} g_i(\beta)$

$= \sum_{i=1}^{n} (\hat{y}_i - y_i) x_i$

$= \sum_{i=1}^{n} -(y_i - m_i \frac{1}{1+e^{-x_i^T \beta}}) x_i$

$= -(y - mw)^T X$

$= \nabla l(\beta)$ (by Exercise 01)

**(B)** Proof:

The probability of picking the $i^{th}$ sample $P(I = i) = \frac{1}{n}$

$E\{g_i(\beta)\} = n \cdot \frac{1}{n} E\{g_i(\beta)\}$

$= \sum_{i=1}^{n} P(i = i) E\{g_I(\beta) | I = i\}$

$= \frac{1}{n} \sum_{i=1}^{n} E\{g_I(\beta) | I = i\}$

$= \frac{1}{n} \nabla l(\beta)$

$\therefore \nabla(\beta) = n E\{g_i(\beta)\}$

**(C)**

**(C - 1) Algorithm**

The algorithm of *Stochastic Gradient Descent* method, is as follows:

(1) Choose an initial vector of parameters $\beta^{(0)}$, and learning rate (step size) $\gamma^{(0)}$;

(2) Repeat until an approximate minimum is obtained:

*(i) Randomly shuffle examples in the training set.

(I noticed that many materials mentioned the shuffling step. I guess this is useful if we do a sample with replacement. Also, since it is not mentioned in the class material, I'm not including the shuffling process in the homework.)

(ii) For $t = 1, 2, 3, ..., n$ do:

$\beta^{(t+1)} = \beta^{(t)} - \gamma^{(t)} g_t(\beta^{(t)})$,

until converged.

**(C - 2) Fixed Step Size**

**(C - 2 - 1)**

We first generate some random samples in order to test-run the algorithm.

Code:

```
###### Generate Random Samples ######

ilogit=function(u) return( 1/(1+exp(-u)));

n=1000
p=5
## create the data matrix
X=cbind(1, matrix(rnorm(n*(p-1)), nrow=n, ncol=(p-1)))

## create the true beta
beta.star=rnorm(p, mean=0, sd=1/sqrt(p))

## have all experiments have an index of 1
m=1
n.list=rep(m, n)

## create the vector of success probabilities
pi.list=ilogit(as.numeric(X%*%beta.star))

## create the vector of observed sample proportions
## of success, one for each of the n Binomial experiments
y = rbinom(n=n, size=n.list, prob=pi.list)


###### Create original beta ######
```

```
fit=glm(y~X[,2:p], family='binomial')
b=1:5


###### Stochastic Gradient Descent Function with Fixed Step Size ######

sgd=function(X, y, stepsize=stepsize, m=1, tol=1e-6, maxit=10000)
{
# note that m is the size of sampled from full dataset

## initialize iteration counter
k=1

iterating=T
bb=matrix(0,p,maxit)
bb[,k]=b
loglikelihood=numeric(maxit)
    pi.t_full=ilogit(as.numeric(X%*%b))
    loglikelihood[k]=sum(n.list*y*log(pi.t_full))+sum(n.list*(1-y)*log(1-pi.t_full))
change=numeric(maxit)
while( iterating )
{
  k=k+1
  # stochastic minibatch
  minibatch=sample(1:n, size=m)
  X.t.n.list.y=X[minibatch,] * n.list[minibatch]*y[minibatch]
  step.size=stepsize
  pi.t=ilogit(as.numeric(X[minibatch,]%*%b))
```

```
  ## negative derivative
  minusGrad=X.t.n.list.y-X[minibatch,] * n.list[minibatch]*pi.t



  add=step.size*minusGrad
  ## advance our iterate
  b=b+add

  ## Calculate Loglikelihood
  pi.t_full=ilogit(as.numeric(X%*%b))
  loglikelihood[k]=sum(n.list*y*log(pi.t_full))+sum(n.list*(1-y)*log(1-pi.t_full))
  bb[,k]=b
  change[k]=sum(bb[k]-bb[k-1])^2
     if( sum(bb[k]-bb[k-1])^2 < tol || (k >=maxit))
    iterating=FALSE

}
#b=as.numeric(b)
b_hat=apply(bb[,(0.5*k+1):k],1,mean)
return(list(b=b, total.iterations=k,beta_tracing=bb[,2:k],loglikelihood=loglikelihood[2:
}

###### Test-run on step size = 0.1 (red) and 0.01 (black) ######

fit_sgd=sgd(X=X, y=y, stepsize=0.1, m=m, maxit=10000)
fit_sgd2=sgd(X=X, y=y, stepsize=0.01, m=m, maxit=10000)

fit_sgd
fit_sgd2
```

```
T=max(fit_sgd$total.iterations-1, fit_sgd2$total.iterations-1)
ts.plot(fit_sgd$loglikelihood[2:T], col='red')
lines(fit_sgd2$loglikelihood[2:T])
```

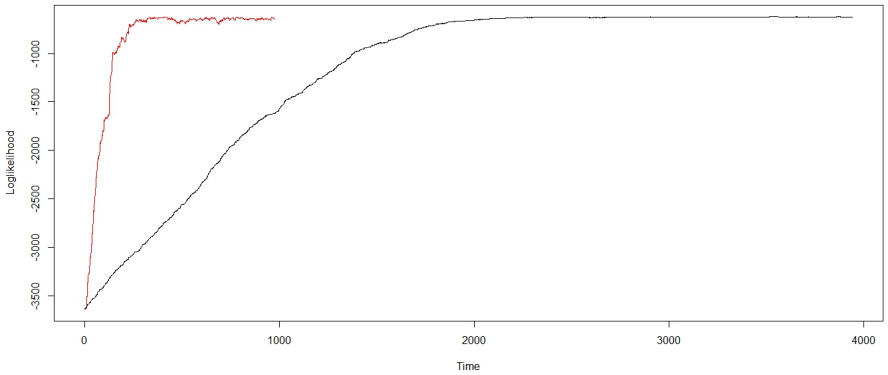


Figure 1: Compare Loglikelihood - SGD Only - Test Dataset - Step Size = 0.1 (Red) vs. 0.01 (Black)

Above is a comparison between two different step sizes on the test dataset, size = 0.1 (in red), vs. size = 0.01 (in black). We can see that a bigger step size leads to a faster convergence and less iteration.

**(C - 2 - 2)**

If we try the same step size on the true dataset, the result is as follows:

The one with smaller step size turns to be more smooth and also converges faster.

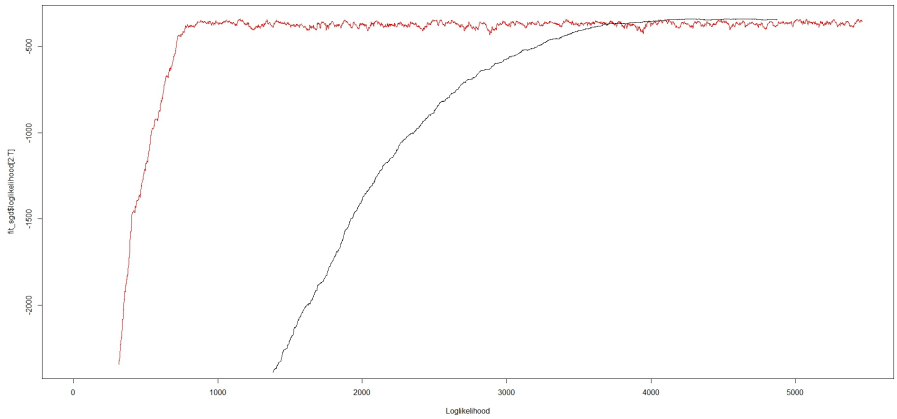


Figure 2: Compare Loglikelihood - SGD Only - True Dataset - Step Size = 0.1 (Red) vs. 0.01 (Black)

### (D) Decaying Step Size

Code:

```
## Create Decaying Step Function

sgd_Decay=function(X, y, t0, C, alpha , m=1, tol=1e-6, maxit=10000)
{
# note that m is the size of minibatch sampled from full dataset

## initialize iteration counter
k=1

iterating=T
bb=matrix(0,p,maxit)
bb[,k]=b
loglikelihood=numeric(maxit)
    pi.t_full=ilogit(as.numeric(X%*%b))
    loglikelihood[k]=sum(n.list*y*log(pi.t_full))+sum(n.list*(1-y)*log(1-pi.t_full))
```

```
change=numeric(maxit)
while( iterating )
{
  k=k+1
  # stochastic minibatch
  minibatch=sample(1:n, size=m)
  X.t.n.list.y=X[minibatch,] * n.list[minibatch]*y[minibatch]
  step.size= C * ( k + t0 )^(- alpha)
  pi.t=ilogit(as.numeric(X[minibatch,]%*%b))

  ## negative derivative of minibatch loss function
  minusGrad=X.t.n.list.y-X[minibatch,] * n.list[minibatch]*pi.t



  add=step.size*minusGrad
  ## advance our iterate
  b=b+add

  ## Calculate Loglikelihood
  pi.t_full=ilogit(as.numeric(X%*%b))
  loglikelihood[k]=sum(n.list*y*log(pi.t_full))+sum(n.list*(1-y)*log(1-pi.t_full))
  bb[,k]=b
  change[k]=sum(bb[k]-bb[k-1])^2
     if( sum(bb[k]-bb[k-1])^2 < tol || (k >=maxit))
    iterating=FALSE
}
#b=as.numeric(b)
b_hat=apply(bb[,(0.5*k+1):k],1,mean)
return(list(b=b, total.iterations=k,beta_tracing=bb[,2:k],loglikelihood=loglikelihood[2:
}
```

```
fit_sgd_Decay=sgd_Decay(X=X, y=y, t0=2, C=10, alpha = 0.6, m=m, maxit=10000)

fit_sgd_Decay2=sgd_Decay(X=X, y=y, t0=2, C=100, alpha = 0.6, m=m, maxit=10000)

fit_sgd_Decay3=sgd_Decay(X=X, y=y, t0=2, C=10, alpha = 0.9, m=m, maxit=10000)

fit_sgd_Decay4=sgd_Decay(X=X, y=y, t0=2, C=100, alpha = 0.9, m=m, maxit=10000)


T=max(fit_sgd_Decay$total.iterations-1, fit_sgd_Decay2$total.iterations-1, fit_sgd_Decay
ts.plot(fit_sgd_Decay$loglikelihood[2:T], col='red', xlab = 'Loglikelihood')
lines(fit_sgd_Decay2$loglikelihood[2:T], col='blue')
lines(fit_sgd_Decay3$loglikelihood[2:T], col='yellow')
lines(fit_sgd_Decay4$loglikelihood[2:T])
```

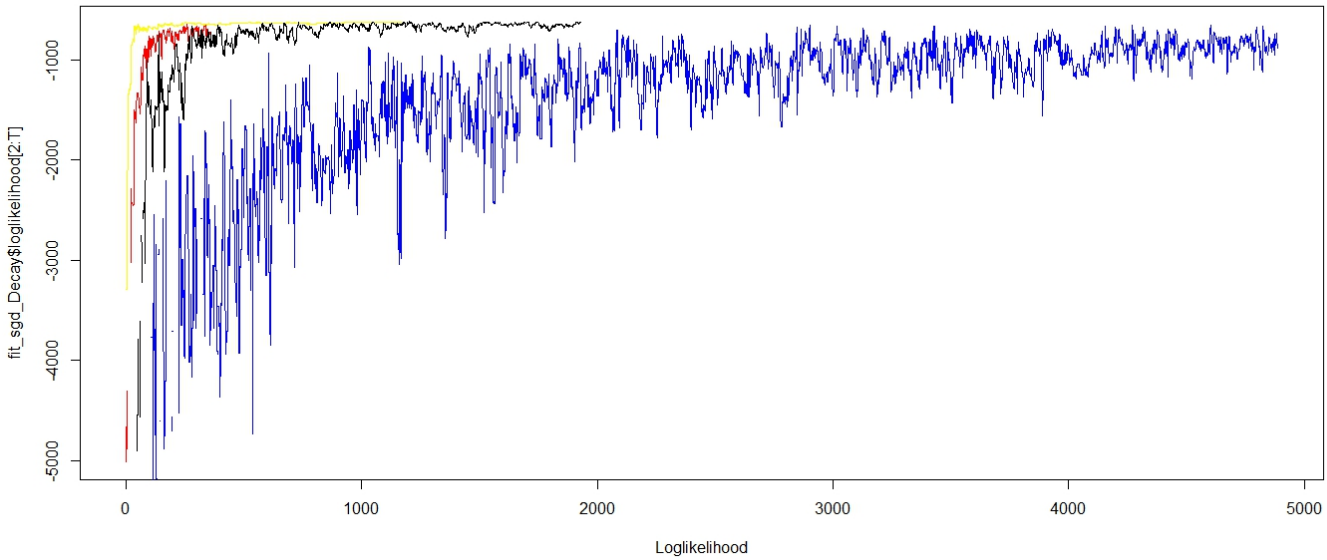


Figure 3: Compare Loglikelihood - SGD Only - Test Dataset - Decaying Step

Above is a comparison between four different decaying step sizes.

$t0 = 2$, $C = 10$, $\alpha = 0.6$ in Red.

$t0 = 2$, $C = 100$, $\alpha = 0.6$ in Blue.

$t0 = 2$, $C = 10$, $\alpha = 0.9$ in Yellow.

$t0 = 2$, $C = 100$, $\alpha = 0.9$ in Black.

We can see that smaller $C$ and $\alpha$ leads to a faster convergence and less iteration.

A bigger $C$ leads to a bigger vibration.

A bigger $\alpha$ leads to a faster convergence.

### (E) Averaging Approach

I dropped the first 100 observation as the burn-in period.

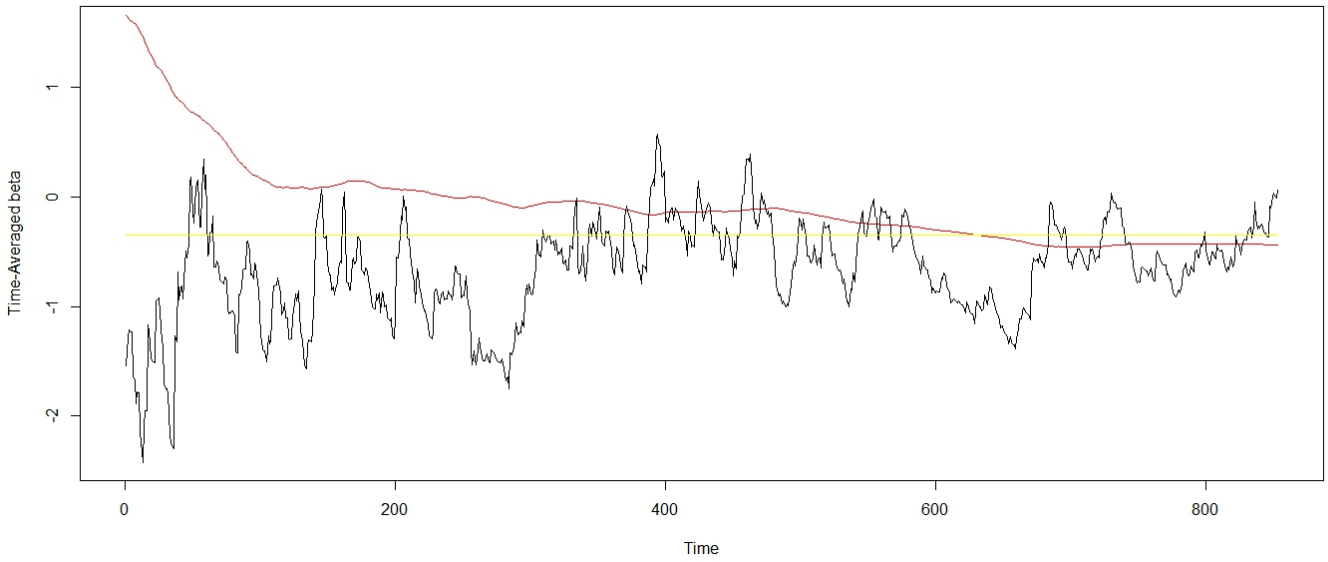


Figure 4: Compare Time-averaged beta (Red) vs. beta (Black)

Compared with the iterates $\beta^{(t)}$, the time-averaged iterates is more smooth and stable, and more closed to the true beta.