Exercises 2 Online Learning - R Code

Natalia Zuniga-Garcia

9/27/2017

Part C

First, I use the functions coded for Exercise 1 Part 2. These functions estimate the succes probability (sigmoid), the negative log-likelihood, and the gradient of the negative log-likelihood.

```
# Function 1. Estimate the Success Probability
wi.estimate <- function(b, X){
  # Inputs:
  # b regression parameter P x 1
  # X Matrix of features N x P
  # Output:
  # wi Sucess Probability N x 1

  w <- 1/(1+exp(-X %*% b))
  w = pmax(w, 1E-6); w = pmin(w, 1-1E-6);
  return(w)
}

# Function 2. Estimate the negative log likelihood
Negll <- function(b, y, X, m){
  # Input:
  # b regression parameter P x 1
  # y vector of response N x 1
  # X Matrix of features N x P
  # m number of trial for the ith case
  # Output:
  # Negative log likelihood of binomial distribution

  ll <- -sum(dbinom(y, m, wi.estimate(b, X), log = TRUE))
  return(ll)
}

# Function 3. Estimate the gradient of the negative log likelihood
Gradll <- function(b, y, X, m){
  # Input:
  # b regression parameter P x 1
  # y vector of response N x 1
  # X Matrix of features N x P
  # m number of trial for the ith case
  # Output:
  # gradient of l(b) P x 1

  grad.ll <- as.numeric(y - m * wi.estimate(b, X)) * X
  return(-colSums(grad.ll))
}
```

Now, I code up the Stochastic Gradient Descent.

```r
# Function 4. Stochastic Gradient Descend
SGradDes <- function(y, X, beta, m, iter, epsilon, alpha){
  # Input:
  # y vector of response N x 1
  # X Matrix of features N x P
  # beta: initial guees of beta P x 1
  # m: number of trial for the ith case
  # iter: maximum iterations allowed if it doesn't converge
  # epsilon: minimum error allowed for convergency creteria
  # alpha: step size (gamma based on class document)
  # Output:
  # Negative log likelihood per iteration
  # Using stochastic gradient descend
  # b regression parameter P x 1

  # Initial Iteration (GUESS)
  betas = array(NA, dim=c(iter, ncol(X)))
  betas[1,] = beta
  ll = array(NA, dim = iter)
  ll[1] = Negll(betas[1,], y, X, m)

  # Iterations
  for (i in 2:iter){
    r <- sample(nrow(X), 1) # Draw a random sample with replacement
    grad <- Gradll(betas[i-1,], matrix(y[r], nrow=1), matrix(X[r,], nrow=1), m)

    # Gradient Descend
    betas[i,] <- betas[i-1,] - alpha * grad
    ll[i] <- Negll(betas[i,], y, X, m)
    # Note that I am tracking the full objective function in every step
    # Thus I am not touching only one point per iteration as SGD intended
    # I'm doing this with the objective of develop a learning process for myself

    # Checking for Convergence
        error = abs((ll[i] - ll[i-1])/(ll[i-1] + epsilon))
    if (error < epsilon){
      cat('Stochastic Gradient Descend has converged in iterations:',i)
      ll <- ll[1:i]
      betas <- betas[1:i,]
      break;
    } else if (i == iter && error >= epsilon){
      print('Stochastic Gradient Descend has not converged')
      break;
    }
  }
  return(list("Negll" = ll, "beta" = betas[i,]))
}
```

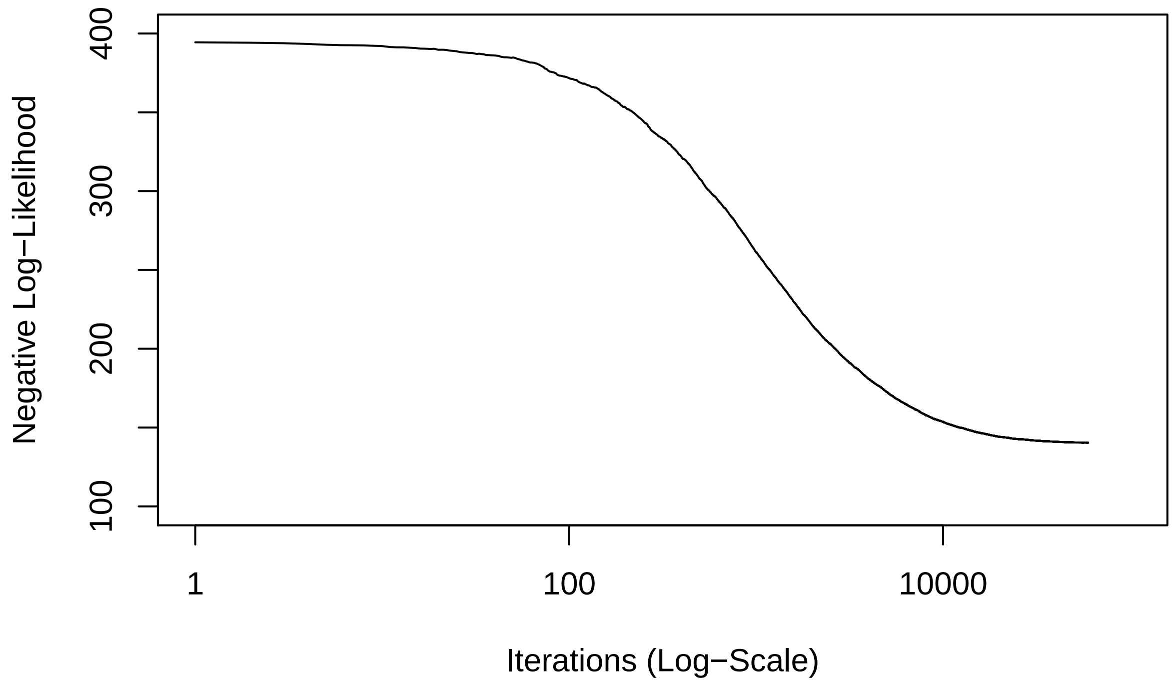
Now I evaluate differente alpha (step size) values using a simulation with wdbc databse.

```r
a <-  c(0.001, 0.01, 0.025, 0.1, 0.25, 1)
for (j in 1:length(a)){
  sgd <-  SGradDes(y, X, beta, m, iter, epsilon, alpha=a[j])
  llSGD <-  as.matrix(unlist(sgd[1], use.names = FALSE))
```

```
  betaSGD <-  as.matrix(unlist(sgd[2], use.names = FALSE))

  #png(filename=paste('F1PC',a[j],'.png'),width=15,height=12,units="cm",res=200)
  plot(1:length(llSGD),llSGD,type="l",col="black", xlab="Iterations (Log-Scale)", ylab="Negative Log-Li
  #dev.off()
}
```

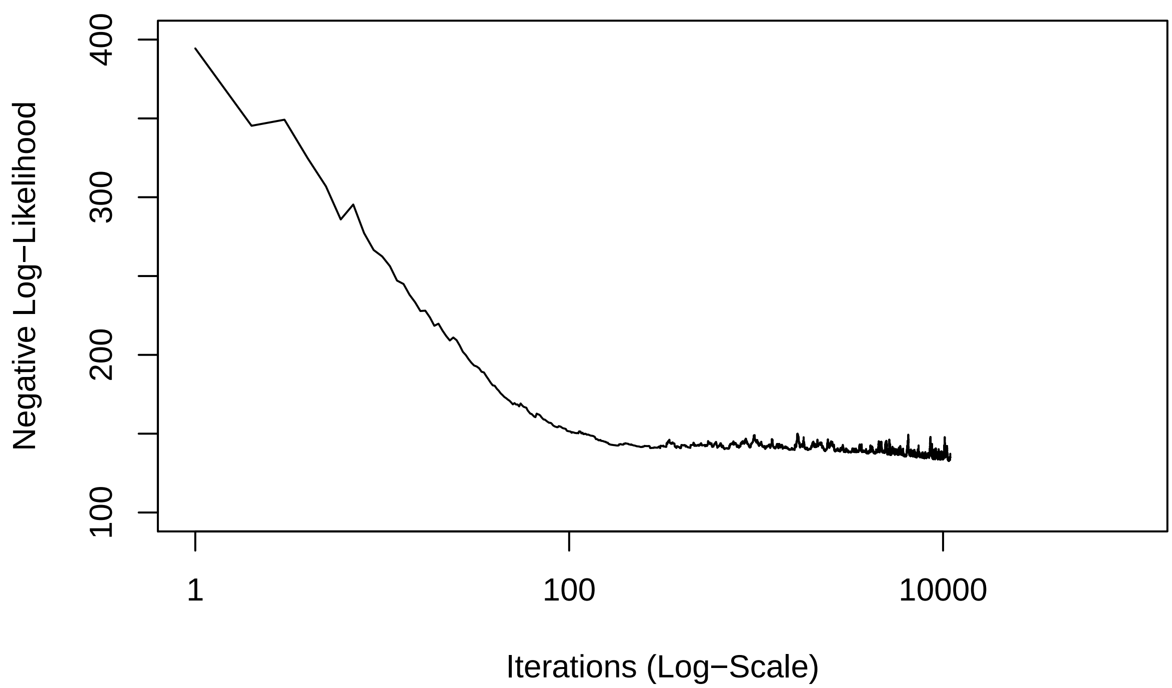
```
## Stochastic Gradient Descend has converged in iterations: 59718
```



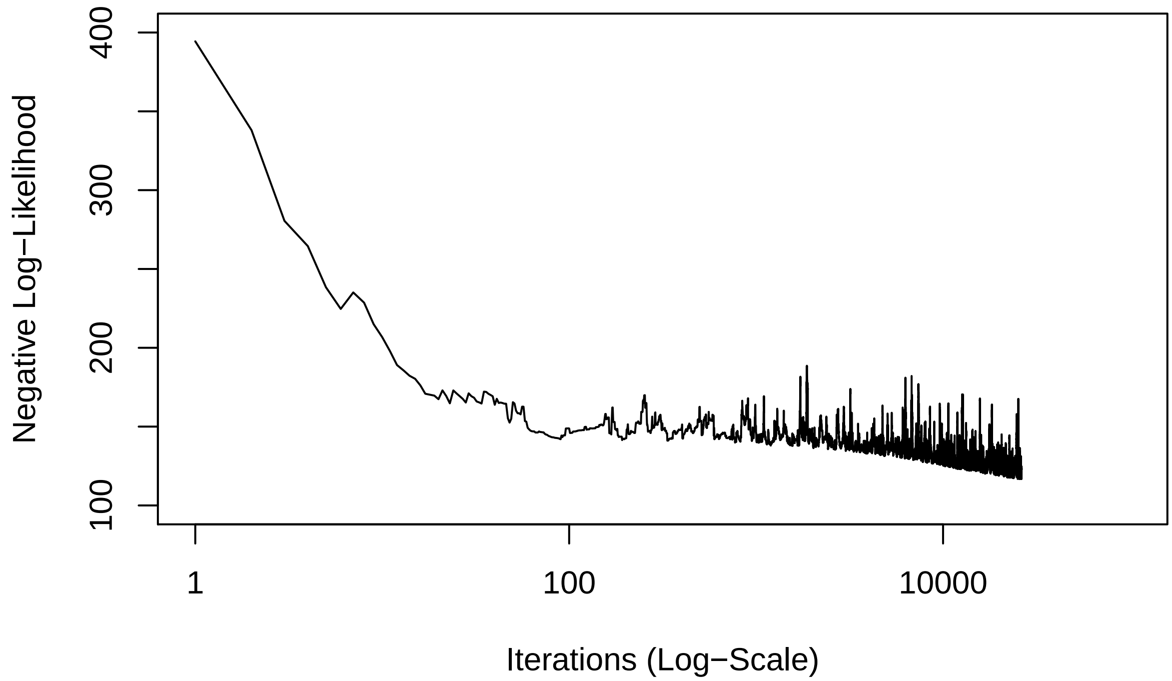

```
## Stochastic Gradient Descend has converged in iterations: 9278
```

```
## Stochastic Gradient Descend has converged in iterations: 9318
```

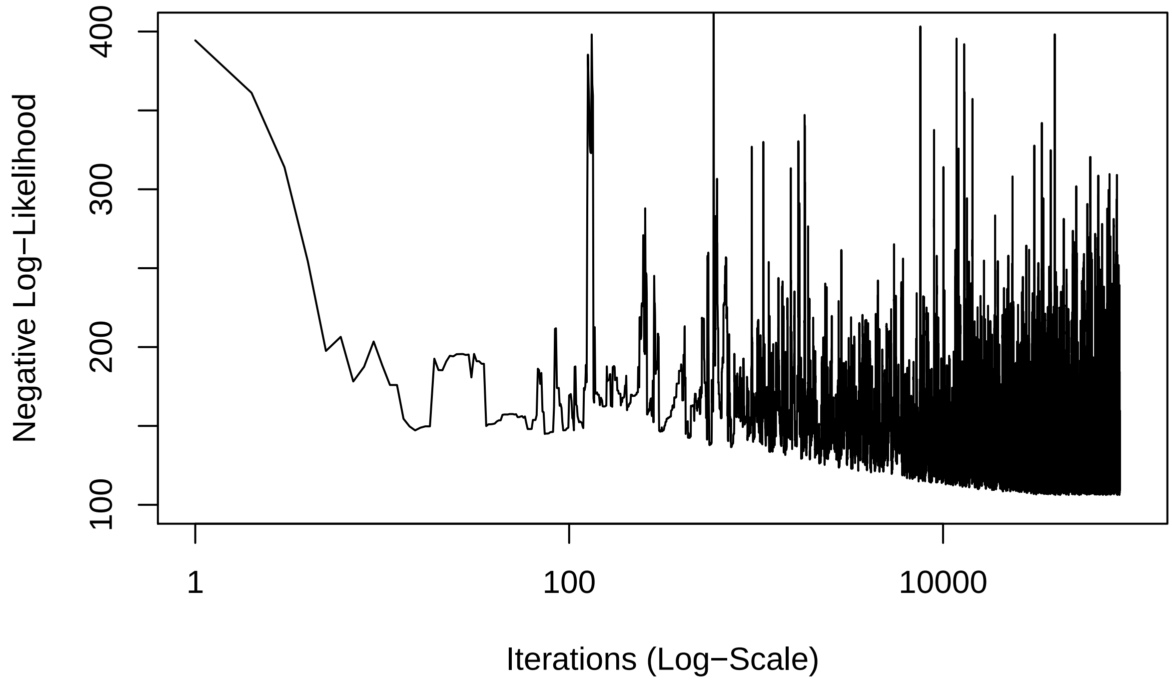

```
## Stochastic Gradient Descend has converged in iterations: 10930
```




```
## Stochastic Gradient Descend has converged in iterations: 26329
```




```
## Stochastic Gradient Descend has converged in iterations: 88220
```



Part D

In this part I implement SGD using a decaying step based on Robbins-Monro rule.

```
# Function 5. Stochastic Gradient Descend Decaying Step Size
SGradDesRM <- function(y, X, beta, m, iter, epsilon, C, t, alpha){
  # Input:
```

```r
  # y vector of response N x 1
  # X Matrix of features N x P
  # beta: initial guees of beta P x 1
  # m: number of trial for the ith case
  # iter: maximum iterations allowed if it doesn't converge
  # epsilon: minimum error allowed for convergency creteria
  # C: constant in the Robbins-Monro rule
  # t: t0 prior number of steps Robbins-Monro rule
  # alpha: learning rate Robbins-Monro rule
  # Output:
  # Negative log likelihood per iteration
  # Using stochastic gradient descend
  # with decaying step size
  # b regression parameter P x 1

  # Initial Iteration (GUESS)
  betas = array(NA, dim=c(iter, ncol(X)))
  betas[1,] = beta
  ll = array(NA, dim = iter)
  ll[1] = Negll(betas[1,], y, X, m)

  # Iterations
  for (i in 2:iter){
    r <- sample(nrow(X), 1) # Draw a random sample with replacement
    grad <- Gradll(betas[i-1,], matrix(y[r], nrow=1), matrix(X[r,], nrow=1), m)

    # Gradient Descend
    step <- C*(i+t)^(-alpha) # Robbins-Monro rule
    betas[i,] <- betas[i-1,] - step * grad
    ll[i] <- Negll(betas[i,], y, X, m)

    # Checking for Convergence
    error = abs((ll[i] - ll[i-1])/(ll[i-1] + epsilon))
    if (error < epsilon){
      cat('Stochastic Gradient Descend has converged in iterations:',i)
      ll <- ll[1:i]
      betas <- betas[1:i,]
      break;
    } else if (i == iter && error >= epsilon){
      print('Stochastic Gradient Descend has not converged')
      break;
    }
  }
  return(list("Negll" = ll, "beta" = betas[i,]))
}
```

Now I evaluate differente C values.

```r
t <- 1
alpha <-  0.6
C <- c(0.01,0.1,0.4,0.6,0.8,1)
for (j in 1:length(a)){
  sgdrm = SGradDesRM(y, X, beta, m, iter, epsilon,C[j], t, alpha)
  llSGDrm = as.matrix(unlist(sgdrm[1], use.names = FALSE))
```

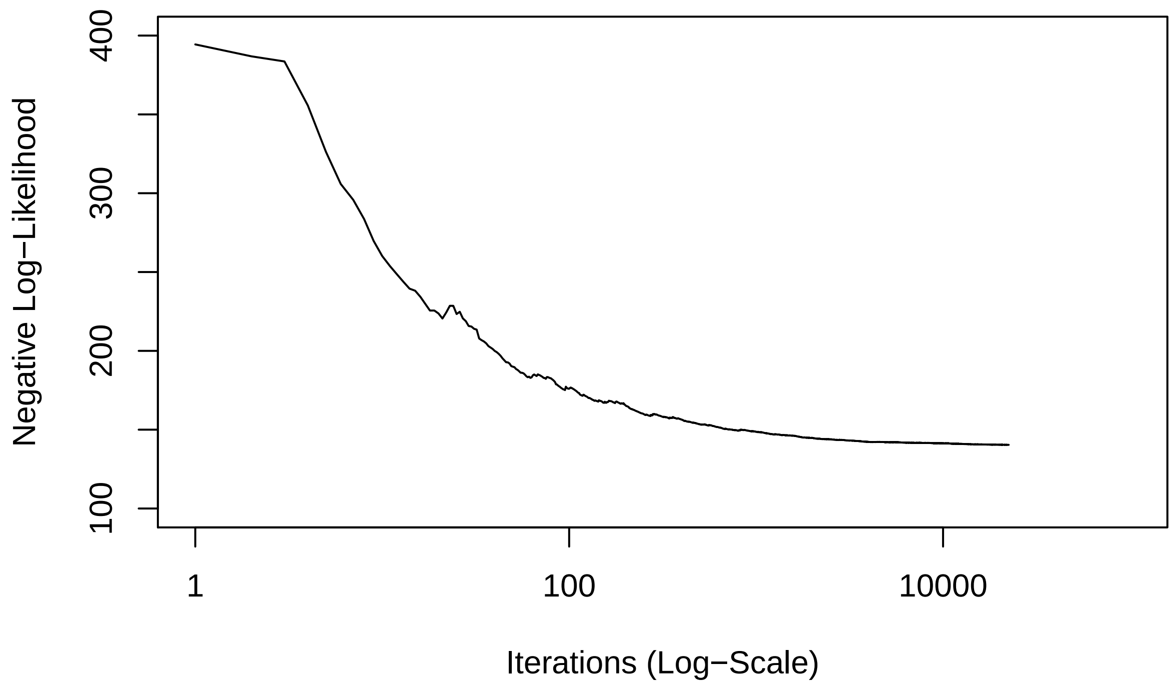
```
  betaSGDrm = as.matrix(unlist(sgdrm[2], use.names = FALSE))

  #png(filename=paste('F2PD',C[j],'.png'),width=15,height=12,units="cm",res=200)
  plot(1:length(llSGDrm),llSGDrm,type="l",col="black", xlab="Iterations (Log-Scale)", ylab="Negative Log
  #dev.off()
}
```

```
## Stochastic Gradient Descend has converged in iterations: 26531
```


```
## Stochastic Gradient Descend has converged in iterations: 57520
```

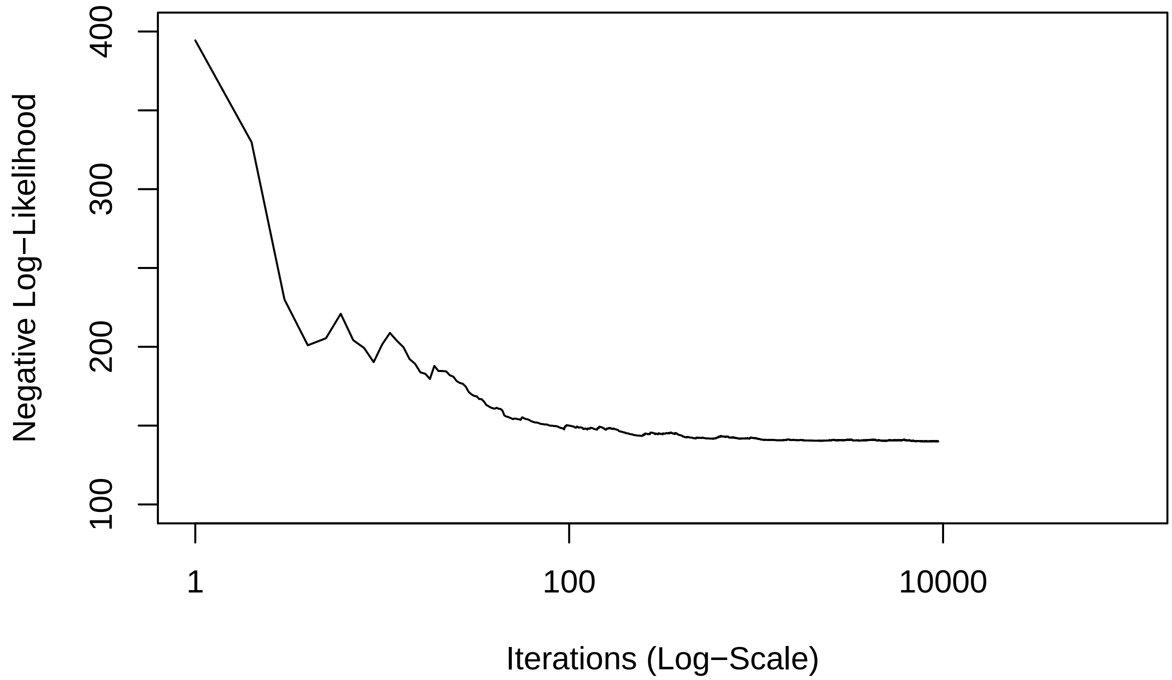
```
## Stochastic Gradient Descend has converged in iterations: 22451
```




```
## Stochastic Gradient Descend has converged in iterations: 14933
```


```
## Stochastic Gradient Descend has converged in iterations: 11887
```


```
## Stochastic Gradient Descend has converged in iterations: 9422
```



Finally I evaluate differente alpha values.

```
C <- 0.1
alpha <-  c(0.5,0.6,0.7,0.8,0.9,1)
for (j in 1:length(alpha)){
  sgdrm = SGradDesRM(y, X, beta, m, iter, epsilon,C, t, alpha[j])
  llSGDrm = as.matrix(unlist(sgdrm[1], use.names = FALSE))
  betaSGDrm = as.matrix(unlist(sgdrm[2], use.names = FALSE))
```

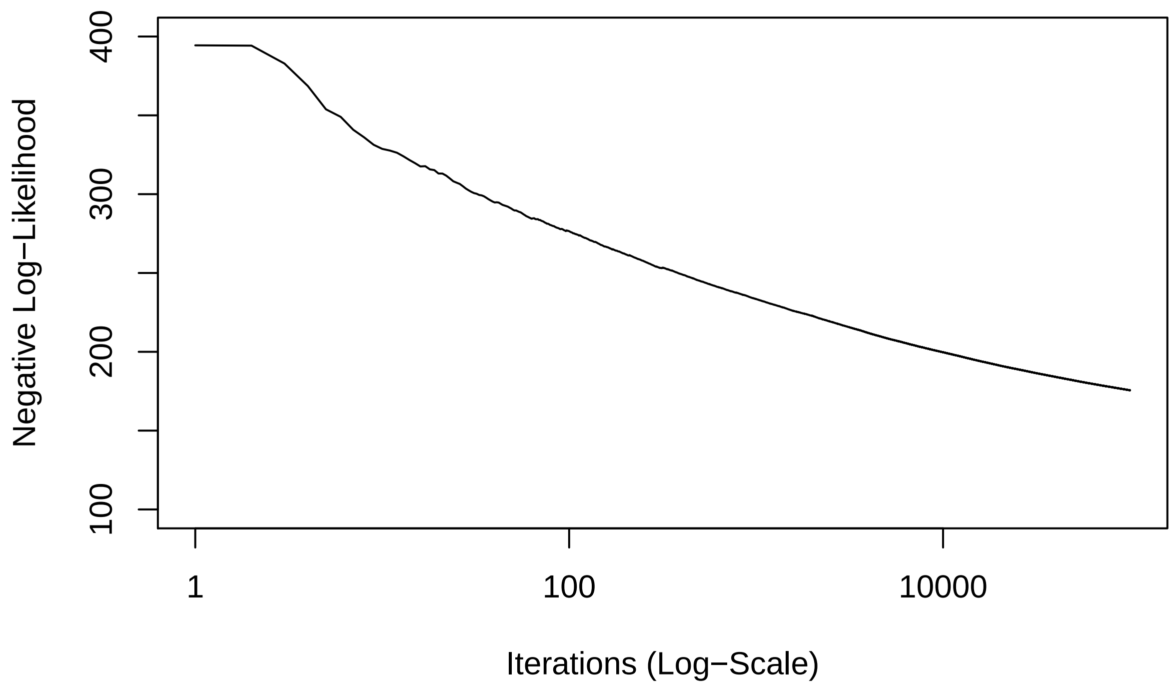
```
  #png(filename=paste('F3PD',alpha[j],'.png'),width=15,height=12,units="cm",res=200)
  plot(1:length(llSGDrm),llSGDrm,type="l",col="black", xlab="Iterations (Log-Scale)", ylab="Negative Log
  #dev.off()
}
```

```
## Stochastic Gradient Descend has converged in iterations: 48745
```


```
## Stochastic Gradient Descend has converged in iterations: 58550
```


```
## [1] "Stochastic Gradient Descend has not converged"
```


```
## [1] "Stochastic Gradient Descend has not converged"
```




```
## Stochastic Gradient Descend has converged in iterations: 64267
```


```
## Stochastic Gradient Descend has converged in iterations: 64288
```