Multinomial Logistic Regression Algorithms

Karl Gregory

10/03/2018

Multinomial logistic regression

Let $Y_1, \ldots, Y_n \in \{1, \ldots, K\}$ be independent random variables and let $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^{d+1}$ be fixed vectors such that

$$Y_i \sim \text{Multinomial}(1, \eta_1(\mathbf{x}_i; \theta), \ldots, \eta_K(\mathbf{x}_i; \theta)),$$

where θ is a parameter vector such that $\theta = (\theta_1^T, \cdots \theta_{K-1}^T)^T$, with $\theta_1, \ldots, \theta_{K-1} \in \mathbb{R}^{d+1}$, and

$$\eta_k(\mathbf{x}; \theta) = e^{\mathbf{x}^T \theta_k}[1 + \sum_{l=1}^{K-1} e^{\mathbf{x}_i^T \theta_l}]^{-1}$$

for $k = 1, \ldots, K-1$ and

$$\eta_K(\mathbf{x}; \theta) = [1 + \sum_{l=1}^{K-1} e^{\mathbf{x}_i^T \theta_l}].$$

The probability mass function of Y_i is thus given by

$$P(Y_i = y) = \prod_{k=1}^{K} [\eta_k(\mathbf{x}_i; \theta)]^{\mathbf{1}(y=k)} \mathbf{1}(y \in \{1, \ldots, K\})$$

for $i = 1, \ldots, n$, so the likelihood function is

$$L(\theta; Y_1, \ldots, Y_n) = \prod_{i=1}^{n} \prod_{k=1}^{K} [\eta_k(\mathbf{x}_i; \theta)]^{\mathbf{1}(y=k)},$$

and the log-likelihood function is

$$\ell(\theta; Y_1, \ldots, Y_n) = \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbf{1}(Y_i = k) \log \eta_k(\mathbf{x}_i; \theta).$$

We do not have a closed form for the maximum likelihood estimator $\hat{\theta}$ for θ, so we must find $\hat{\theta}$ numerically. We consider three algorithms: Coordinate descent, a ridge-stabilized Newton-Raphson algorithm, and a fixed-Hessian Newton-Raphson algorithm. In developing these algorithms, we will make use of the fact that

$$\frac{\partial}{\partial \theta_k} \eta_l(\mathbf{x}; \theta) = \eta_l(\mathbf{x}; \theta)[\mathbf{1}(k = l) - \eta_k(\mathbf{x}; \theta)]\mathbf{x}$$

for all $1 \leq k, l \leq K$, which can be shown without too much trouble. Moreover we have

$$\frac{\partial}{\partial \theta_k} \ell(\theta; Y_1, \ldots, Y_n) = \sum_{i=1}^{n} [\mathbf{1}(Y_i = k) - \eta_k(\mathbf{x}_i; \theta)]\mathbf{x}_i$$

for $k = 1, \ldots, K-1$ and

$$\frac{\partial^2}{\partial \theta_k \partial \theta_l} \ell(\theta; Y_1, \ldots, Y_n) = -\sum_{i=1}^{n} \eta_k(\mathbf{x}_i; \theta)[\mathbf{1}(Y_i = l) - \eta_l(\mathbf{x}_i; \theta)]\mathbf{x}_i \mathbf{x}_i^T$$

for $1 \leq k, l \leq K-1$, from which we define the $(K-1)(d+1) \times 1$ score vector and the $(K-1)(d+1) \times (K-1)(d+1)$ Hessian matrix as

$$S(\theta) = \left(\sum_{i=1}^{n} [\mathbf{1}(Y_i = k) - \eta_k(\mathbf{x}_i; \theta)] \mathbf{x}_i \right)_{1 \leq k \leq K-1}$$

and

$$H(\theta) = \left(-\sum_{i=1}^{n} \eta_k(\mathbf{x}_i; \theta)[\mathbf{1}(k = l) - \eta_l(\mathbf{x}_i; \theta)] \mathbf{x}_i \mathbf{x}_i^T \right)_{1 \leq k,l \leq K-1}.$$

Coordinate descent

The idea of coordinate descent algorithms is to iteratively maximize a function with respect to one parameter at a time, holding the others fixed. One cycles repeatedly through the parameters until convergence. Precisely, in our case, given initial values $\theta_1^-, \ldots, \theta_{K-1}^-$, coordinate descent prescribes repeating the following until convergence:

Outer loop:

Update θ_1^- to θ_1^+ by

$$\theta_1^+ \leftarrow \text{argmax}_{\theta_1} \, \ell(\theta_1, \theta_2^-, \ldots, \theta_{K-1}^-; Y_1, \ldots, Y_n),$$

and then for $k = 2, \ldots, K-2$, update θ_k^- to θ_k^+ by

$$\theta_k^+ \leftarrow \text{argmax}_{\theta_k} \, \ell(\theta_1^+, \ldots, \theta_{k-1}^+, \theta_k, \theta_{k+1}^-, \ldots, \theta_{K-1}^-; Y_1, \ldots, Y_n),$$

and then update θ_{K-1}^- to θ_{K-1}^+ by

$$\theta_{K-1}^+ \leftarrow \text{argmax}_{\theta_k} \, \ell(\theta_1^+, \ldots, \theta_{K-2}^+, \theta_{K-1}; Y_1, \ldots, Y_n).$$

Each of the maximizations may be carried out using a Newton-Raphson algorithm. For any $k = 1, \ldots, K-1$, the Newton-Raphson updates are as follows:

Inner loop:

For initial values $\theta_1^*, \ldots, \theta_{K-1}^*$, update θ_k^* to θ_k^{**} by

$$\theta_k^{**} \leftarrow \theta_k^* - H_k^{-1}(\theta^*) S_k(\theta^*),$$

where

$$S_k(\theta) = \sum_{i=1}^{n} [\mathbf{1}(Y_i = k) - \eta_k(\mathbf{x}_i; \theta)] \mathbf{x}_i$$

and

$$H_{k,k}(\theta) = -\sum_{i=1}^{n} \eta_k(\mathbf{x}_i; \theta)[1 - \eta_k(\mathbf{x}_i; \theta)] \mathbf{x}_i \mathbf{x}_i^T.$$

These updates may be reformulated such that θ_k^{**} is the solution to a least squares problem: If we define the $n \times n$ matrix

$$W_k(\theta) = -\text{diag}(\eta_k(\mathbf{x}_1; \theta)[1 - \eta_k(\mathbf{x}_1; \theta)], \ldots, \eta_k(\mathbf{x}_n; \theta)[1 - \eta_k(\mathbf{x}_n; \theta)])$$

and the $n \times 1$ vector

$$U_k(\theta) = (\mathbf{1}(Y_1 = k) - \eta_k(\mathbf{x}_1; \theta), \ldots, \mathbf{1}(Y_n = k) - \eta_k(\mathbf{x}_n; \theta))^T,$$

then we may write

$$H_k(\theta) = -\mathbf{X}^T W_k(\theta)\mathbf{X} \quad \text{and} \quad S_k(\theta) = \mathbf{X}^T U_k(\theta).$$

Then we may express the update as

$$\theta_k^{**} = \theta_k^* + (\mathbf{X}^T W_k(\theta^*)\mathbf{X})^{-1}\mathbf{X}^T U_k(\theta^*) = (\mathbf{X}^T W_k(\theta^*)\mathbf{X})^{-1}\mathbf{X}^T W_k(\theta^*)[\mathbf{X}\theta_k^* + W^{-1}(\theta^*)U_k(\theta^*)].$$

Letting

$$Z_k(\theta^*) = \mathbf{X}\theta_k^* + W^{-1}(\theta^*)U_k(\theta^*),$$

we have

$$\theta_k^* - H_k^{-1}(\theta^*)S_k(\theta^*) = (\mathbf{X}^T W_k(\theta^*)\mathbf{X})^{-1}\mathbf{X}^T W_k(\theta^*)Z_k(\theta^*).$$

Finally, defining

$$\tilde{Z}_k^* = W_k^{1/2}(\theta^*)Z_k(\theta^*) \quad \text{and} \quad \tilde{\mathbf{X}}_k^* = W_k^{1/2}(\theta^*)\mathbf{X},$$

we may write

$$\theta_k^{**} = (\tilde{\mathbf{X}}_k^{*T}\tilde{\mathbf{X}}_k^*)^{-1}\tilde{\mathbf{X}}_k^*\tilde{Z}_k^* = \text{argmin}_\theta \|\tilde{Z}_k^* - \tilde{\mathbf{X}}_k^*\theta\|_2^2.$$

A good reference for this method is Friedman, Hastie, and Tibshirani (2010). The code below implements the coordinate descent algorithm for computing the maximum likelihood estimator of θ.

```
# In all the R code, there are K sets of parameters so that
# theta = (theta_1,...,theta_K), but
# for identifiability, theta_K = 0 always.

# define the softmax function
softmax <- function(x){exp(x)/sum(exp(x))}

# generate some multinomial data:
n <- 50
d <- 2
K <- 3
XX <- cbind(rep(1,n),matrix(rnorm(n*d),n,d))
THETA <- cbind(matrix(rnorm((d+1)*(K-1)),d+1,K-1),rep(0,d+1))
ETA <- t(apply(XX %*% THETA,1,softmax))
Y <- apply(ETA,1,FUN=sample, size=1,x=1:K, replace=FALSE)

# initialize parameter values
THETA.1 <- matrix(0,d+1,K)
ETA.1 <- t(apply(XX %*% THETA.1,1,softmax))
ETA.0 <- 1 # let the loop begin

# set convergence criterion
delta <- .001
iter <- 0

while( max(abs(ETA.0 - ETA.1 )) > delta) # stop when fitted probabilities cease to change.
{

  ETA.0 <- ETA.1
  THETA.0 <- THETA.1

  for( k in 1:(K-1))
  {

    THETA.0[,k] <- THETA.1[,k] + 1  # let the loop begin
```

```
    while( max(abs(THETA.1[,k] - THETA.0[,k])) > delta)
    {

      THETA.0[,k] <- THETA.1[,k]

      p.k <- t(apply(XX %*% THETA.0,1,softmax))[,k]
      w.k <- p.k*(1-p.k)
      Z.k <- XX %*% THETA.0[,k] + ( (Y == k) - p.k) / w.k
      W.k <- diag(w.k)
      XX.w.k <- sqrt(W.k) %*% XX
      Z.w.k <- sqrt(w.k) * Z.k
      theta.1.k <- solve(t(XX.w.k) %*% XX.w.k) %*% t(XX.w.k) %*% Z.w.k

      THETA.1[,k] <- theta.1.k

    }
  }

  ETA.1 <- t(apply(XX %*% THETA.1,1,softmax))

  iter <- iter + 1
}

THETA.1.CD <- THETA.1

print(iter)
```

```
## [1] 8
```

```
print(THETA.1.CD)
```

```
##            [,1]      [,2] [,3]
## [1,]  0.1429447 2.0140096    0
## [2,] -0.5358643 0.8258555    0
## [3,]  0.6997993 1.7411876    0
```

These results are very close (try playing with the convergence criterion) to those from the package `nnet`, as can be seen after some re-parameterizing:

```
library(nnet)
# relabel Y so that we get the same answer from the multinom function,
# which uses a different identifiability constraint:
Y.multinom <- -Y
multinom.coefs <- as.matrix(summary(multinom(Y.multinom ~ XX[,-1]))$coefficients)
```

```
## # weights:  12 (6 variable)
## initial  value 54.930614
## iter  10 value 31.325856
## final  value 31.308472
## converged
```

```
if(K==2){ multinom.coefs <- t(multinom.coefs)}
THETA.1.nnet <- t(as.matrix(rbind(multinom.coefs[(K-1):1,],rep(0,ncol(XX)))))
print(THETA.1.nnet)
```

```
##                    -1        -2
```

```
## (Intercept)  0.1500389 2.0181569 0
## XX[, -1]1   -0.5356763 0.8252182 0
## XX[, -1]2    0.7040395 1.7437920 0
```

Ridge-stabilized Newton-Raphson

Given an initial value θ^- of the parameter θ, the Newtown-Raphson algorithm prescribes the updates

$$\theta^+ \leftarrow \theta^- - H^{-1}(\theta^-)S(\theta^-).$$

This algorithm, however, is frought with convergence issues for $K \geq 3$. Sometimes the Hessian will not be negative definite, and sometimes the quadratic approximation upon which each Newton-Raphson update is based is very poor. The following algorithm, taken from Goldfeld, Quandt, and Trotter (1966), addresses these problems in each iteration.

Let λ_1 be the greatest eigenvalue of $H(\theta^-)$ and let R be a constant which will play the role of controlling the step size of the algorithm. Then the ridge-stabilized Newton-Raphson algorithm proceeds via the updates

$$\theta^+ \leftarrow \theta^- - H_\alpha^{-1}(\theta^-)S(\theta^-),$$

where, for $\alpha = \lambda_1 + R\|S(\theta^-)\|_2$,

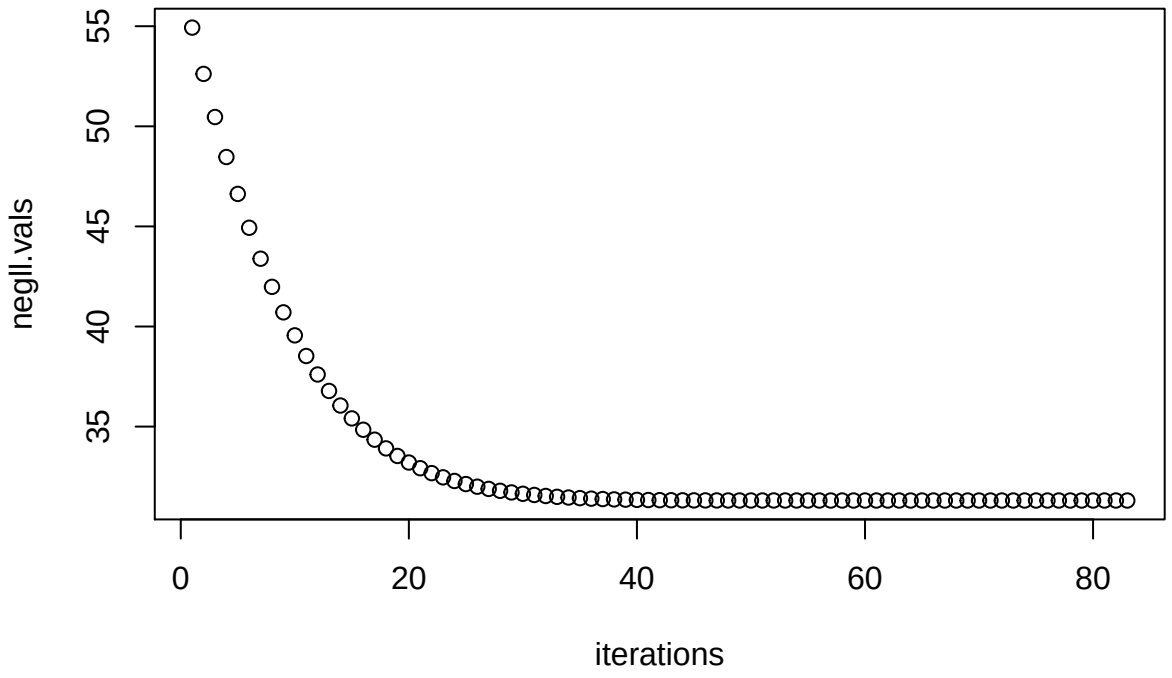
$$H_\alpha(\theta^-) = \begin{cases} H(\theta^-) - \alpha I & \text{if } \alpha > 0 \\ H(\theta^-) & \text{if } \alpha \leq 0. \end{cases}$$

The modified Hessian $H_\alpha(\theta^-)$ is always negative definite. Goldfeld, Quandt, and Trotter (1966) prescribe a way to adjust the constant R in each iteration. This is implemented in the code below, which carries out the algorithm.

```
# define function to evaluate negative log-likelihood
negll <- function(THETA,Y,XX)
{

    K <- max(Y)
    negll <- 0
    PROBS <- t(apply(XX %*% THETA,1,softmax))
    for( k in 1:K){

        negll <- negll - sum( (Y==k)*log(PROBS[,k]) )

    }
    return(negll)

}

# initialize parameter values
THETA.1 <- matrix(0,d+1,K)
ETA.1 <- t(apply(XX %*% THETA.1,1,softmax))
ETA.0 <- 1 # let the loop begin

# initialize iteration counter and create empty vector in which to store
# evaluations of the negative log-likelihood function after every step
iter <- 1
negll.vals <- numeric()
negll.vals[1] <- negll(THETA.1,Y,XX)
```

```
R <- 10 # set initial value for "radius", which is kind of like a step size.
SS <- 1 # allow algorithm to begin
while(max(abs(SS)) > delta)
{

    ETA.0 <- ETA.1
    THETA.0 <- THETA.1

    ETA.1 <- t(apply(XX %*% THETA.0,1,softmax))

    # Build Hession HH and score function SS

    HH <- matrix(NA,(K-1)*(d+1),(K-1)*(d+1))
    SS <- numeric((K-1)*(d+1))

    for(k1 in 1:(K-1))
    {

        ind1 <- ((k1-1)*(d+1) + 1):(k1*(d+1))
        SS[ind1] <- t(XX) %*% ((Y == k1) - ETA.1[,k1])

        for(k2 in 1:(K-1))
        {

            ind2 <- ((k2-1)*(d+1) + 1):(k2*(d+1))

            if(k1 == k2)
            {

                HH[ind1,ind2] <-   - t(XX) %*% diag(ETA.1[,k1]*(1 - ETA.1[,k1])) %*% XX

            } else {

                HH[ind1,ind2] <-  - t(XX) %*% diag(ETA.1[,k1]*ETA.1[,k2]) %*% XX

            }
        }
    }

    HH[is.na(HH)] <- 0

    # compute alpha
    lambda.1 <- max(eigen(HH)$values)
    norm.SS <- sqrt(sum(SS^2))
    alpha <- lambda.1 + R * norm.SS

    # compute modified Hessian
    BB <- HH - (alpha > 0) * alpha * diag((K-1)*(d+1))

    # get update
    Theta.0.long <- as.vector(THETA.0[,-K])
    Theta.1.long <- Theta.0.long - solve(BB) %*% SS # do update with modified Hessian
```

```
    THETA.1 <- cbind(matrix(Theta.1.long,ncol=K-1),rep(0,d+1))
    ETA.1 <- t(apply(XX %*% THETA.1,1,softmax))

  # adjust R
  dTheta <- Theta.1.long - Theta.0.long
  negll.LQA <- - (negll.vals[iter] + SS %*% dTheta + (1/2)*t(dTheta) %*% HH %*% dTheta)
  negll.vals[iter+1] <- negll(THETA.1,Y,XX)
  delta.negll <- negll.vals[iter+1] - negll.vals[iter]
  delta.negll.LQA <- negll.LQA - negll.vals[iter]

  z <- delta.negll / delta.negll.LQA

  if( z < 0){

    R <- R * 4

    } else if((.7 < z) & (z < 1.3)){

    R <- R * 0.4

    } else if(z > 2){

    R <- R * 4

    }

    iter <- iter + 1

}

plot(negll.vals, xlab="iterations")
```



```
THETA.1.ridgestabilizedNR <- THETA.1

print(THETA.1.ridgestabilizedNR)
```

```
##             [,1]      [,2] [,3]
## [1,]  0.1496101 2.0176628    0
## [2,] -0.5358097 0.8253049    0
## [3,]  0.7037386 1.7435231    0
```

We see that the results are the same.

Fixed-Hessian algorithm

A disadvantage of the Newton-Raphson algorithm is that the Hessian must be inverted in every iteration. This incurs a high computational cost. Böhning (1992) proposed an algorithm (see the paper for complete details) which replaces the inverse of the Hessian in all iterations with a single fixed matrix. As a result, the algorithm tends to require more iterations, but each iteration is very cheap. Given an initial parameter value θ^-, Böhning (1992) prescribes the updates

$$\theta^+ \leftarrow \theta^- - B^{-1}S(\theta^-),$$

where

$$B = -(K/2) \times (K^{-1}\mathbf{I} - K^{-2}\mathbf{1}\mathbf{1}^T) \otimes \mathbf{X}^T\mathbf{X},$$

where $\mathbf{I}$ is the $(K-1) \times (K-1)$ identity matrix and $\mathbf{1}$ is a $(K-1) \times 1$ vector of ones. This gives

$$B^{-1} = -2(\mathbf{I} + \mathbf{1}\mathbf{1}^T) \otimes (\mathbf{X}^T\mathbf{X})^{-1}.$$

Note that the matrix

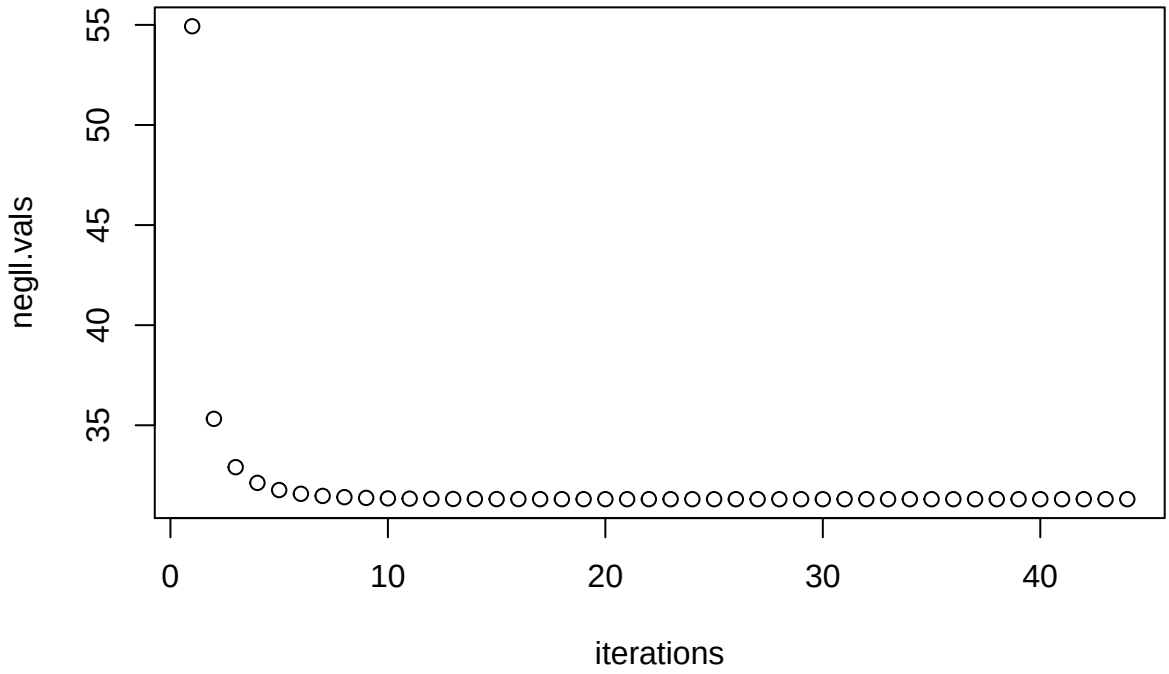
$$-(K^{-1}\mathbf{I} - K^{-2}\mathbf{1}\mathbf{1}^T) \otimes \mathbf{X}^T\mathbf{X}$$

is the Hessian under the uniform Multinomial distribution, that is, when $\eta_1(\mathbf{x}_i;\theta) = \cdots = \eta_K(\mathbf{x}_i;\theta) = 1/K$ for $i = 1, \ldots, n$. See Böhning (1992) for further details.

The advantage of this algorithm is that we only need to compute the matrix B^{-1} once. The code below implements the algorithm:

```
# initialize parameter values
THETA.1 <- matrix(0,d+1,K)
ETA.1 <- t(apply(XX %*% THETA.1,1,softmax))
ETA.0 <- 1 # let the loop begin

iter <- 1
negll.vals <- numeric()
negll.vals[1] <- negll(THETA.1,Y,XX)

# Build B inverse:
B.inv <- - 2 * (diag(K-1) + matrix(1,K-1,K-1)) %x% solve( t(XX) %*% XX )

# allow algorithm to begin
SS <- 1
while(max(abs(SS)) > delta)
{

    ETA.0 <- ETA.1
    THETA.0 <- THETA.1
```

```
    ETA.1 <- t(apply(XX %*% THETA.0,1,softmax))

    # Build score function SS

    SS <- numeric((K-1)*(d+1))

    for(k1 in 1:(K-1))
    {

        ind1 <- ((k1-1)*(d+1) + 1):(k1*(d+1))

        SS[ind1] <- t(XX) %*% ((Y == k1) - ETA.1[,k1])

    }

    # Perform update
    Theta.0.long <- as.vector(THETA.0[,-K])
    Theta.1.long <- Theta.0.long - B.inv %*% SS

    THETA.1 <- cbind(matrix(Theta.1.long,ncol=K-1),rep(0,d+1))
    ETA.1 <- t(apply(XX %*% THETA.1,1,softmax))

    negll.vals[iter+1] <- negll(THETA.1,Y,XX)

    iter <- iter + 1

}

plot(negll.vals,xlab="iterations")
```




```
THETA.1.fixedHNR <- THETA.1
```

```r
print(THETA.1.fixedHNR)
```

```
##            [,1]      [,2] [,3]
## [1,]  0.1506317 2.0183429    0
## [2,] -0.5351952 0.8253456    0
## [3,]  0.7042880 1.7439444    0
```

References

Böhning, Dankmar. 1992. "Multinomial Logistic Regression Algorithm." *Annals of the Institute of Statistical Mathematics* 44 (1). Springer: 197–200.

Friedman, Jerome, Trevor Hastie, and Rob Tibshirani. 2010. "Regularization Paths for Generalized Linear Models via Coordinate Descent." *Journal of Statistical Software* 33 (1). NIH Public Access: 1.

Goldfeld, Stephen M, Richard E Quandt, and Hale F Trotter. 1966. "Maximization by Quadratic Hill-Climbing." *Econometrica: Journal of the Econometric Society*. JSTOR, 541–51.