SimulationAG1AG2PoissonGLM_Weighted_Gradient

Xiangni Peng

```r
library(dplyr)
```

```r
set.seed(1028)

# Define the number of simulation
num_sim <- 100
```

Poisson GLM setup

```r
# Generate the Poisson data: the design matrix X and counts y
generate_data <- function(N,beta_true,sigma_c){

  # p dimension of the parameters
  p = length(beta_true)

  # Generate the covariate matrix
  C <- matrix(rnorm(N*(p-1),mean = 0, sd = sigma_c), nrow = N, ncol = p-1)

  # Add the intercept to get the design matrix X
  X <- cbind(1,C)

  # Find the linear predictor
  linear_predictor <- as.numeric(X %*% beta_true)

  # Find the mean (lambda)
  lambda_value <- exp(linear_predictor)

  # Generate the counts y
  y <- rpois(N, lambda = lambda_value)

  return(list(X=X,y=y))
}
```

Gradient in Poisson GLM

For each client i with samples $\{(x_{ij}, y_{ij})\}_{j=1}^{n}$, assume $Y_{ij} \sim \text{Poisson}(\lambda)$ with the log link:

$$\log(\lambda) = x_{ij}^{\top}\beta.$$

The negative log-likelihood is

$$\rho(x_{ij}, y_{ij}; \beta) = \exp(x_{ij}^{\top}\beta) - y_{ij}\, x_{ij}^{\top}\beta,$$

where $\log(y_{ij}!)$ is omitted since it does not depend on β.

The gradient of this loss is

$$\nabla\rho(x_{ij}, y_{ij}; \beta) = x_{ij}\big(\exp(x_{ij}^\top\beta) - y_{ij}\big).$$

Therefore, the local loss for client i is

$$L_i(\beta) = \frac{1}{n}\sum_{j=1}^{n}\rho(x_{ij}, y_{ij}; \beta) = \frac{1}{n}\sum_{j=1}^{n}\Big(\exp(x_{ij}^\top\beta) - y_{ij}\,x_{ij}^\top\beta\Big),$$

and the corresponding local gradient is

$$\nabla L_i(\beta) = \frac{1}{n}\,X_i^\top\big(\exp(X_i\beta) - y_i\big),$$

with the global gradient given by

$$\nabla L(\beta) = \frac{1}{m}\sum_{i=1}^{m}\nabla L_i(\beta).$$

```r
# Find the gradient of rho for a single point (one data point in a client)
gradient_rho_single <- function(x_ij,y_ij,beta){
  lambda_ij <- exp(sum(x_ij*beta))
  g <- x_ij*(lambda_ij - y_ij)
  return(as.numeric(g))
}
# Find its norm
gradient_rho_single_norm <- function(x_ij, y_ij, beta) {
  lambda_ij <- exp(sum(x_ij * beta))
  g <- x_ij * (lambda_ij - y_ij)
  return(sqrt(sum(g^2)))  # L2 norm
}

# Gradient Clipping
clipped_gradient <- function(g,B){
  norm_g <- sqrt(sum(g^2))
  if (norm_g > B){
    return(g*(B/norm_g))
  } else {
    return(g)
  }
}
```

Federated Setup: Split the pooled dataset into m clients

```r
# Split the pooled dataset into m clients with their corresponding sample size
split_client <- function(X,y,client_ni){
  # client_ni is a vector that include the local sample size for each client.
  # Assign the first n1 rows to client 1;
  # then the n1+1 to n1+n2 th row to client 2,...,until client m

  # Extract the starting row for each client
  starts <- c(1, head(cumsum(client_ni)+1,-1))

  # Extract the ending row for each client
  ends <- cumsum(client_ni)
  # Store all the clients into a list.
  # For i=1,...,m, client i has its design matrix Xi and response yi
  client_list <- vector("list", length(client_ni))
  for (i in 1:length(client_ni)){
    client_list[[i]] <- list(X = X[starts[i]:ends[i], , drop = FALSE],
                             y = y[starts[i]:ends[i]])
  }

  return(client_list)
}
```

Other Function(s):

```
# Calculate the mse of beta_hat
mse_beta <- function(beta_hat, beta_true) {
  return(mean((beta_hat - beta_true)^2))
}
```

Other Notations

```
# Define the true parameters
beta_true <- c(1,2,-1,3)

# p dimension of the parameters
p <- length(beta_true)

# Covariate matrix (used for generating the design matrix X)
sigma_c <- 0.2

# Privacy budget
mu <- 2
```

Algorithm 1: K-Iteration Federated M-Estimator

```
# K is the number of iterations
K <- 50

eta <- 0.5

# B will be defined as the 90th percentile of norms of the gradients
```

```
# Define \beta^{k+1} in Algorithm 1
update_beta_AG1 <- function(last_beta,eta,mu,client_Bi,K,client_list,client_ni){

  p = length(last_beta)
  m = length(client_list)
```

```r
  # Define gradient matrix which is a m X p matrix
  private_gradient_matrix <- matrix(NA_real_, nrow = m, ncol = p)

  # Step 2 in Algorithm 1
  for (i in 1:m) {
    ni <- client_ni[i]
    # a.(i) Client i computes its gradients for the all the local data
    all_local_gradients_at_client_i <- matrix(NA_real_,nrow = ni, ncol = p)
    # Calculate the gradient for each data point in client i
    for (j in 1:ni){
      X_ij <- client_list[[i]]$X[j, ]
      y_ij <- client_list[[i]]$y[j]
      g_ij <- gradient_rho_single(X_ij, y_ij, last_beta)
      all_local_gradients_at_client_i[j, ] <- clipped_gradient(g_ij, client_Bi[i])
    }
    avg_gi = colMeans(all_local_gradients_at_client_i)

    sum_ni_square = sqrt(sum(client_ni^2))

    # a.(ii)Set the private scale
    mult_i <- 2*client_Bi[i]*sqrt(K)/(mu*sum_ni_square)
    # Find noise_i for client i
    Zi <- rnorm(p)
    noise_i <- mult_i*Zi

    # a.(iii) Client i sends the privatized gradients
    private_gradient_matrix[i, ] <- avg_gi+noise_i
  }

    # b. Server takes the weighted average of all private gradient sent by each client
    weight_ni <- client_ni/sum(client_ni)
    g_tilde <- as.numeric(t(weight_ni) %*% private_gradient_matrix)

    # c. update beta
    new_beta <- last_beta - eta*g_tilde
    return(as.numeric(new_beta))
}
```

```r
# Define \beta^{k+1} in Algorithm 1
update_beta_AG1_not_weighted <- function(last_beta,eta,mu,client_Bi,K,client_list,client_ni){

  p = length(last_beta)
  m = length(client_list)


  # Define gradient matrix which is a m X p matrix
  private_gradient_matrix <- matrix(NA_real_, nrow = m, ncol = p)

  # Step 2 in Algorithm 1
  for (i in 1:m) {
    ni <- client_ni[i]
    # a.(i) Client i computes its gradients for the all the local data
    all_local_gradients_at_client_i <- matrix(NA_real_,nrow = ni, ncol = p)
    # Calculate the gradient for each data point in client i
    for (j in 1:ni){
      X_ij <- client_list[[i]]$X[j, ]
      y_ij <- client_list[[i]]$y[j]
      g_ij <- gradient_rho_single(X_ij, y_ij, last_beta)
      all_local_gradients_at_client_i[j, ] <- clipped_gradient(g_ij, client_Bi[i])
    }
    avg_gi = colMeans(all_local_gradients_at_client_i)

    # a.(ii)Set the private scale
    mult_i <- 2*client_Bi[i]*sqrt(K)/(mu*ni*sqrt(m))
    # Find noise_i for client i
    Zi <- rnorm(p)
    noise_i <- mult_i*Zi

    # a.(iii) Client i sends the privatized gradients
    private_gradient_matrix[i, ] <- avg_gi+noise_i
  }

    # b. Server takes the average of all private gradient sent by each client
    g_tilde <- colMeans(private_gradient_matrix)

    # c. update beta
    new_beta <- last_beta - eta*g_tilde
    return(as.numeric(new_beta))
}
```

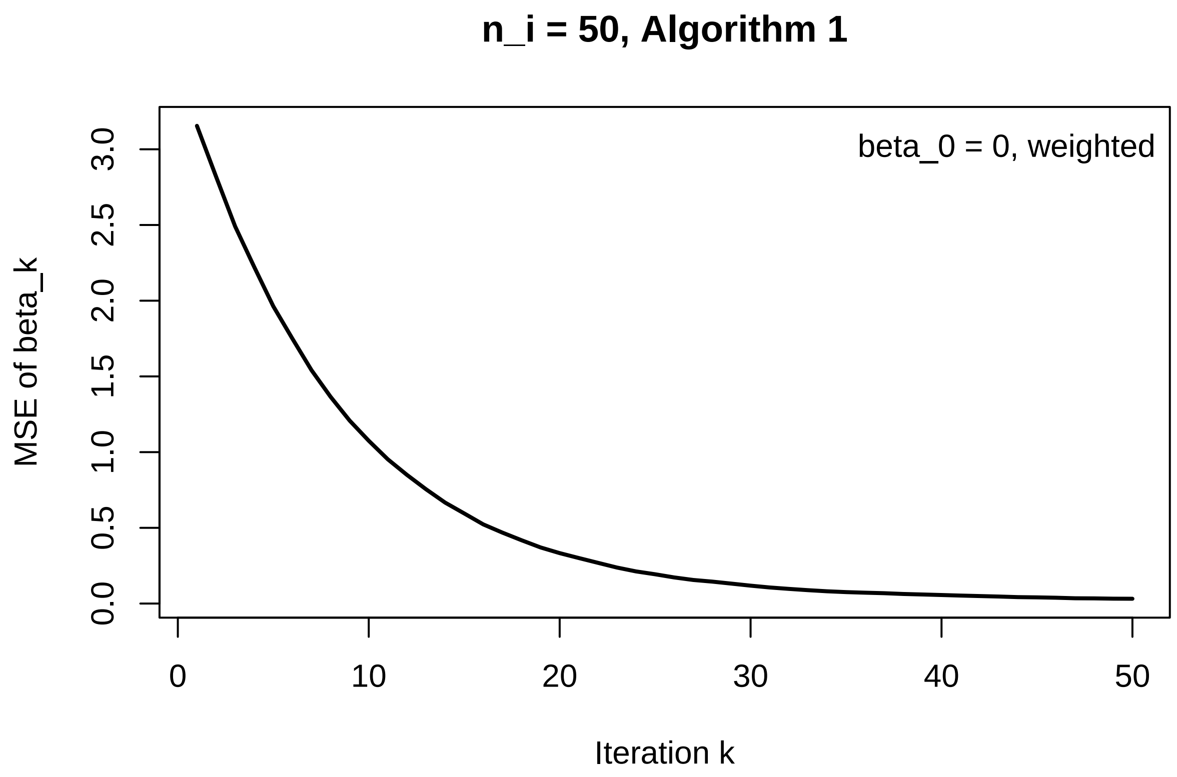
1.1 all clients have the same local sample size

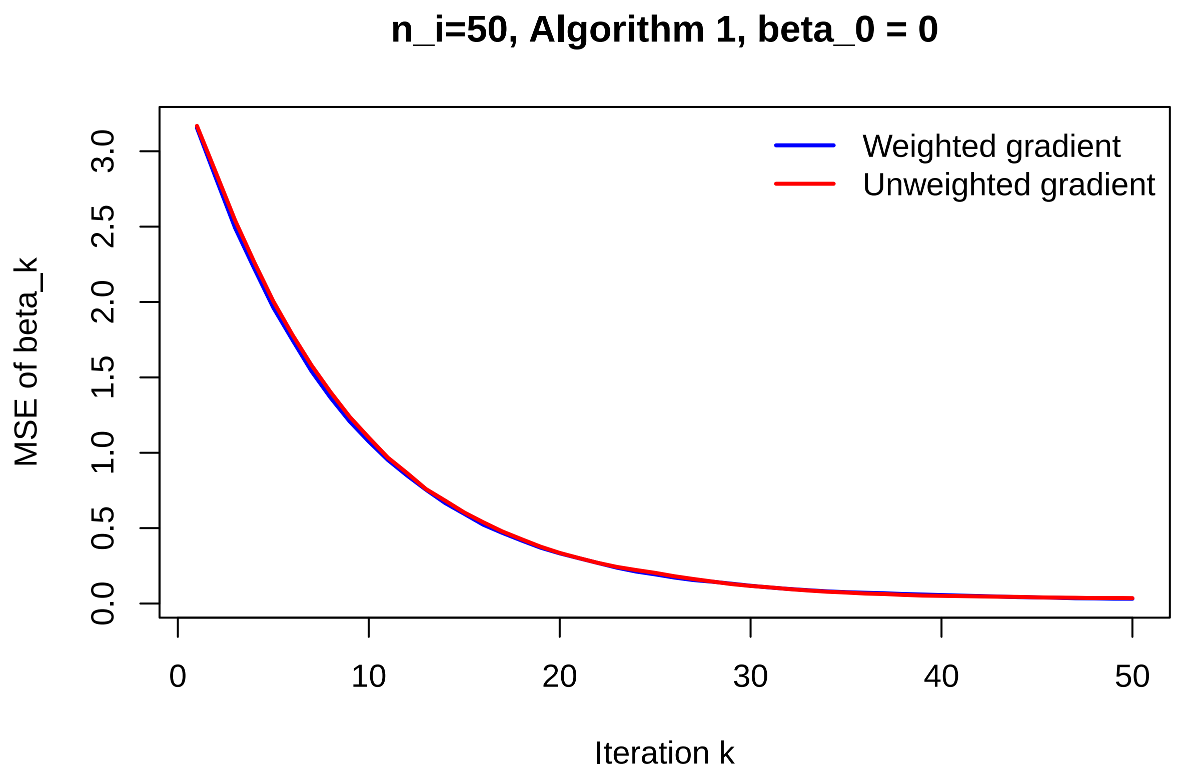
```r
client_ni_11 <- c(50,50,50,50,50,50,50,50,50,50)
N11 <- sum(client_ni_11)
```

1.1.1 initial value is zero

```r
# Create an initial matrix to store all the simulation results
sim_results_mse_mat111 <- matrix(NA_real_,nrow = num_sim,ncol = K)

for (sim_i in 1:num_sim){
  # Set the initial value beta_0 as 0
  beta_0 <- rep(0,p)
  beta_path <- matrix(NA_real_,nrow = K+1, ncol = p)
  beta_path[1, ] <- beta_0
  mse_path <- rep(NA_real_,K+1)

  # Generate data and split them into m clients
  data_sim_i <- generate_data(N11, beta_true, sigma_c)
  clients_sim_i <- split_client(data_sim_i$X, data_sim_i$y, client_ni_11)
  m = length(clients_sim_i)

  # Step 1
  # 1.(i)Find Bi
  client_Bi <- rep(NA_real_,m)
  for (i in 1:m) {
    all_local_gradient_norm_at_client_i <- rep(NA_real_,client_ni_11[i])
    # Calculate the norm of each local gradient
    for (j in 1:client_ni_11[i]){
      X_ij <- clients_sim_i[[i]]$X[j, ]
      y_ij <- clients_sim_i[[i]]$y[j]
      g_norm_ij <- gradient_rho_single_norm(X_ij, y_ij, beta_0)
      all_local_gradient_norm_at_client_i[j] <- g_norm_ij
    }
    # Let B be the 90th percentile of the norms
    Bi <- as.numeric(quantile(all_local_gradient_norm_at_client_i, probs = 0.9))
    client_Bi[i] <- Bi
  }

  for (k in 1:K){
    beta_path[k+1, ] <- update_beta_AG1(as.numeric(beta_path[k,]),eta,
                                         mu,client_Bi,K,clients_sim_i,client_ni_11)
    mse_path[k+1] <- mse_beta(as.numeric(beta_path[k+1,]),beta_true)
  }

  # Store the results
  sim_results_mse_mat111[sim_i, ] <- mse_path[-1]
}
```

```r
# Plot mse vs k
plot_mse_path <- colMeans(sim_results_mse_mat111)
plot(1:K, plot_mse_path, type = "l", lwd = 2,
     xlab = "Iteration k",
     ylab = "MSE of beta_k",
     main = "n_i = 50, Algorithm 1")
legend("topright", legend = c("beta_0 = 0, weighted"), bty = "n")
```


n_i = 50, Algorithm 1
beta_0 = 0, weighted
MSE of beta_k
Iteration k
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
10
20
30
40
50



1.2 local sample sizes are not the same

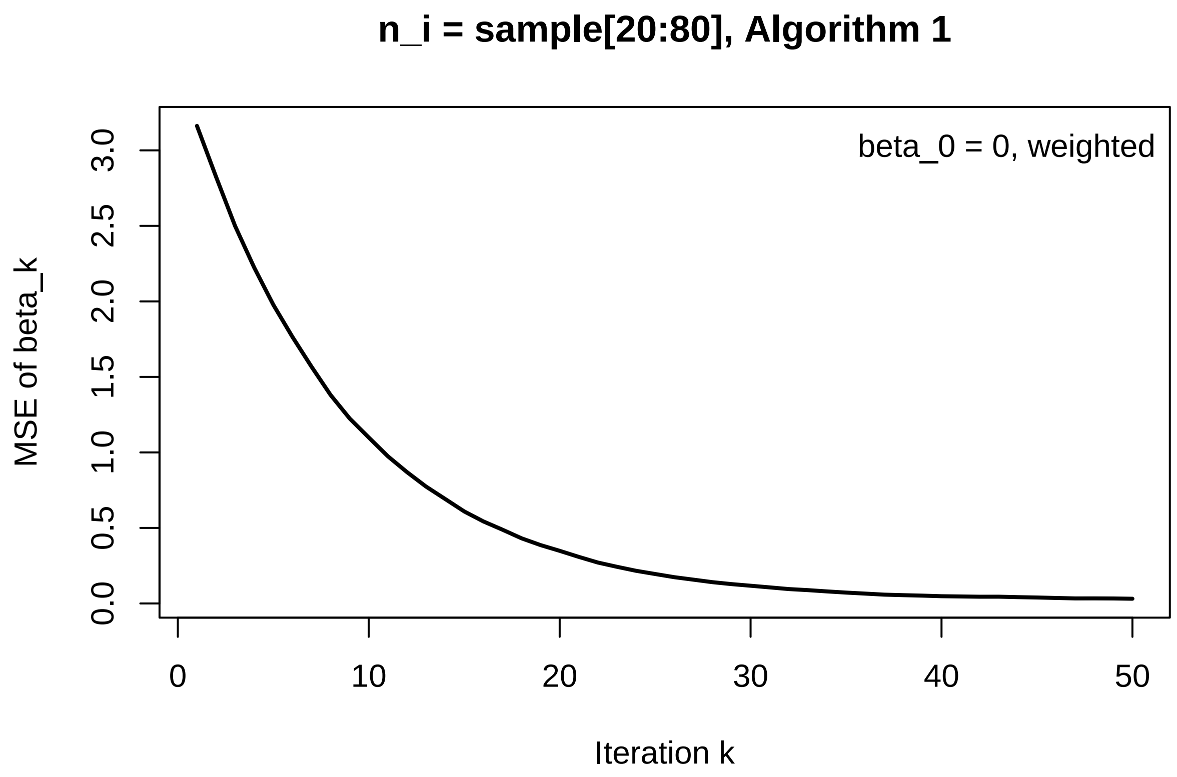
```
m = 10
client_ni_12 <- sample(20:80,m,replace = TRUE)
N12 <- sum(client_ni_12)
```

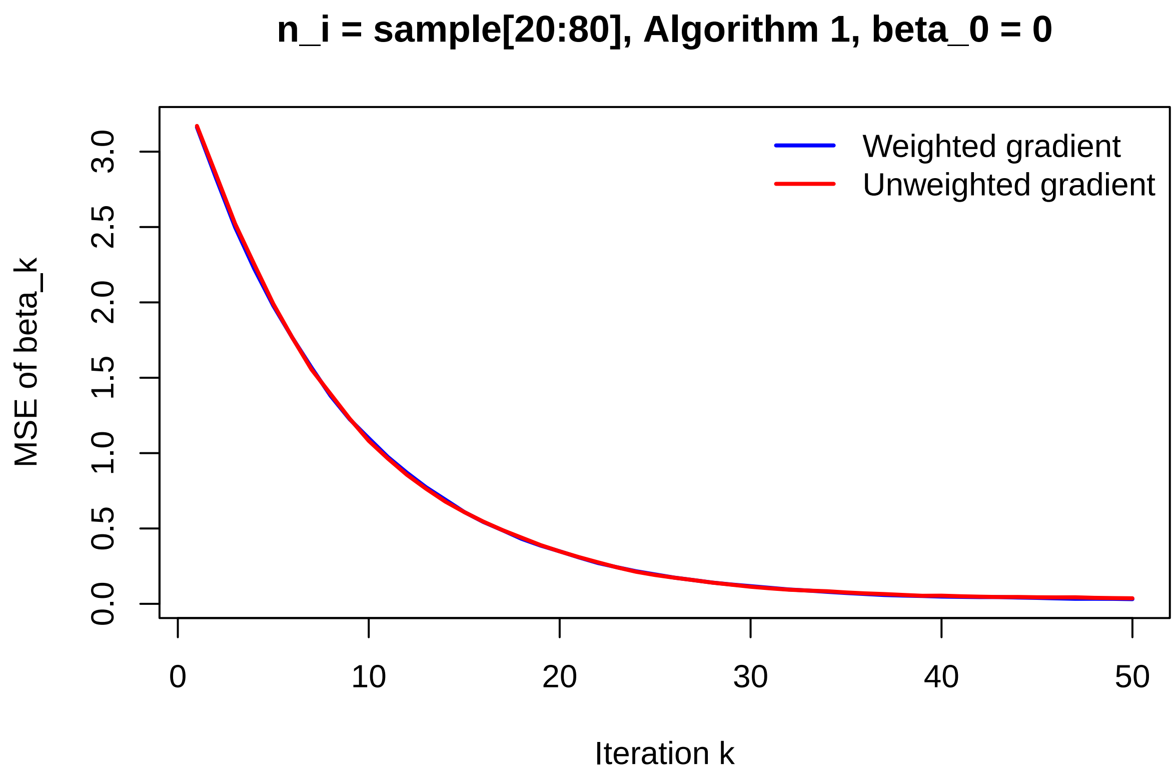
1.2.1 start with zero

```
# Create an initial matrix to store all the simulation results
sim_results_mse_mat121 <- matrix(NA_real_,nrow = num_sim,ncol = K)

for (sim_i in 1:num_sim){
  # Set the initial value beta_0 as 0
  beta_0 <- rep(0,p)
  beta_path <- matrix(NA_real_,nrow = K+1, ncol = p)
  beta_path[1, ] <- beta_0
  mse_path <- rep(NA_real_,K+1)

  # Generate data and split them into m clients
  data_sim_i <- generate_data(N12, beta_true, sigma_c)
  clients_sim_i <- split_client(data_sim_i$X, data_sim_i$y, client_ni_12)
```

```r
  m = length(clients_sim_i)

  # Find Bi
  client_Bi <- rep(NA_real_,m)
  for (i in 1:m) {
    all_local_gradient_norm_at_client_i <- rep(NA_real_,client_ni_12[i])
    # Calculate the norm of each local gradient
    for (j in 1:client_ni_12[i]){
      X_ij <- clients_sim_i[[i]]$X[j, ]
      y_ij <- clients_sim_i[[i]]$y[j]
      g_norm_ij <- gradient_rho_single_norm(X_ij, y_ij, beta_0)
      all_local_gradient_norm_at_client_i[j] <- g_norm_ij
    }
    # Let B be the 90th percentile of the norms
    Bi <- as.numeric(quantile(all_local_gradient_norm_at_client_i, probs = 0.9))
    client_Bi[i] <- Bi
  }

  for (k in 1:K){
    beta_path[k+1, ] <- update_beta_AG1(as.numeric(beta_path[k,]),eta,
                                   mu,client_Bi,K,clients_sim_i,client_ni_12)
    mse_path[k+1] <- mse_beta(as.numeric(beta_path[k+1,]),beta_true)
  }

  # Store the results
  sim_results_mse_mat121[sim_i, ] <- mse_path[-1]
}
```

```r
# Plot mse vs k
plot_mse_path121 <- colMeans(sim_results_mse_mat121)
plot(1:K, plot_mse_path121, type = "l", lwd = 2,
     xlab = "Iteration k",
     ylab = "MSE of beta_k",
     main = "n_i = sample[20:80], Algorithm 1")
legend("topright", legend = c("beta_0 = 0, weighted"), bty = "n")
```


n_i = sample[20:80], Algorithm 1
beta_0 = 0, weighted
MSE of beta_k
Iteration k
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
10
20
30
40
50



Algorithm 1 applied in stage II of Algorithm 2

```r
# Define \beta^{k+1} in Algorithm 1 applied in stage II of Algorithm 2
update_beta_AG1_in_AG2 <- function(last_beta,eta,mu,client_Bi,K,client_list,client_ni){

  p = length(last_beta)
  m = length(client_list)

  # Define gradient matrix which is a m X p matrix
  private_gradient_matrix <- matrix(NA_real_, nrow = m, ncol = p)

  # Step 2 in Algorithm 1
  for (i in 1:m) {
    ni <- client_ni[i]
    # a.(i) Client i computes its gradients for the all the local data
    all_local_gradients_at_client_i <- matrix(NA_real_,nrow = ni, ncol = p)
    # Calculate the gradient for each data point in client i
    for (j in 1:ni){
      X_ij <- client_list[[i]]$X[j, ]
      y_ij <- client_list[[i]]$y[j]
      g_ij <- gradient_rho_single(X_ij, y_ij, last_beta)
      all_local_gradients_at_client_i[j, ] <- clipped_gradient(g_ij, client_Bi[i])
    }
    avg_gi = colMeans(all_local_gradients_at_client_i)
    sum_ni_square = sqrt(sum(client_ni^2))

    # a.(ii)Set the private scale
    mult_i <- 2*client_Bi[i]*sqrt(2*K)/(mu*sum_ni_square)
    # Find noise_i for client i
    Zi <- rnorm(p)
    noise_i <- mult_i*Zi

    # a.(iii) Client i sends the privatized gradients
    private_gradient_matrix[i, ] <- avg_gi+noise_i
  }

    # b. Server takes the weighted average of all private
    # gradient sent by each client
    weight_ni <- client_ni/sum(client_ni)
    g_tilde <- as.numeric(t(weight_ni) %*% private_gradient_matrix)

    # c. update beta
    new_beta <- last_beta - eta*g_tilde
    return(as.numeric(new_beta))
}
```

```
AG1_in_AG2 <- function(client_list, client_ni, K2, eta2, mu, beta_0) {

  m <- length(client_list)
  p <- length(beta_0)

  # Set the initial value
  beta_path <- matrix(NA_real_, nrow = K2+1, ncol = p)
  beta_path[1, ] <- beta_0

  # Find Bi
  client_Bi <- rep(NA_real_,m)
  for (i in 1:m) {
    all_local_gradient_norm_at_client_i <- rep(NA_real_,client_ni[i])
    # Calculate the norm of each local gradient
    for (j in 1:client_ni[i]){
      X_ij <- client_list[[i]]$X[j, ]
      y_ij <- client_list[[i]]$y[j]
      g_norm_ij <- gradient_rho_single_norm(X_ij, y_ij, beta_0)
      all_local_gradient_norm_at_client_i[j] <- g_norm_ij
    }
    # Let B be the 90th percentile of the norms
    Bi <- as.numeric(quantile(all_local_gradient_norm_at_client_i, probs = 0.9))
    client_Bi[i] <- Bi
  }

  # K2 iterations
    for (k in 1:K2){
    beta_path[k+1, ] <- update_beta_AG1_in_AG2(as.numeric(beta_path[k,]),eta2,
                                       mu,client_Bi,K2,client_list,client_ni)
  }
  return(beta_path)
}
```

2. Algorithm 2: K-Local / K-Server Federated M-Estimator

```r
eta_1 = 0.2
K1 = 50
m = 10
mu = 2
eta_2 = 0.5
K2 = 50
```

```r
# For one client, update the beta in K1 iterations
update_local_beta_AG2_stageI <- function(last_beta, eta_1, mu,
                                         B, K1, X_i, y_i,m){



  p = length(last_beta)
  n = length(y_i)

  # Find the gradient for every data point in client i
  all_local_gradients_at_client_i <- matrix(NA_real_,nrow = n, ncol = p)
    # Calculate the norm of each local gradient
    for (j in 1:n){
      X_ij <- X_i[j, ]
      y_ij <- y_i[j]
      g_ij <- gradient_rho_single(X_ij, y_ij, last_beta)
      all_local_gradients_at_client_i[j, ] <- clipped_gradient(g_ij, B)
    }
  g_tilde <- colMeans(all_local_gradients_at_client_i)

  # noise
  # Set privacy scales
  a = 2*B*eta_1*sqrt(2*m*K1)/(mu*n)
  Z0 <- rnorm(p,0,1)
  noise <- a*Z0

  # update beta
  new_beta <- last_beta - eta_1*g_tilde + noise
  return(as.numeric(new_beta))
}
```

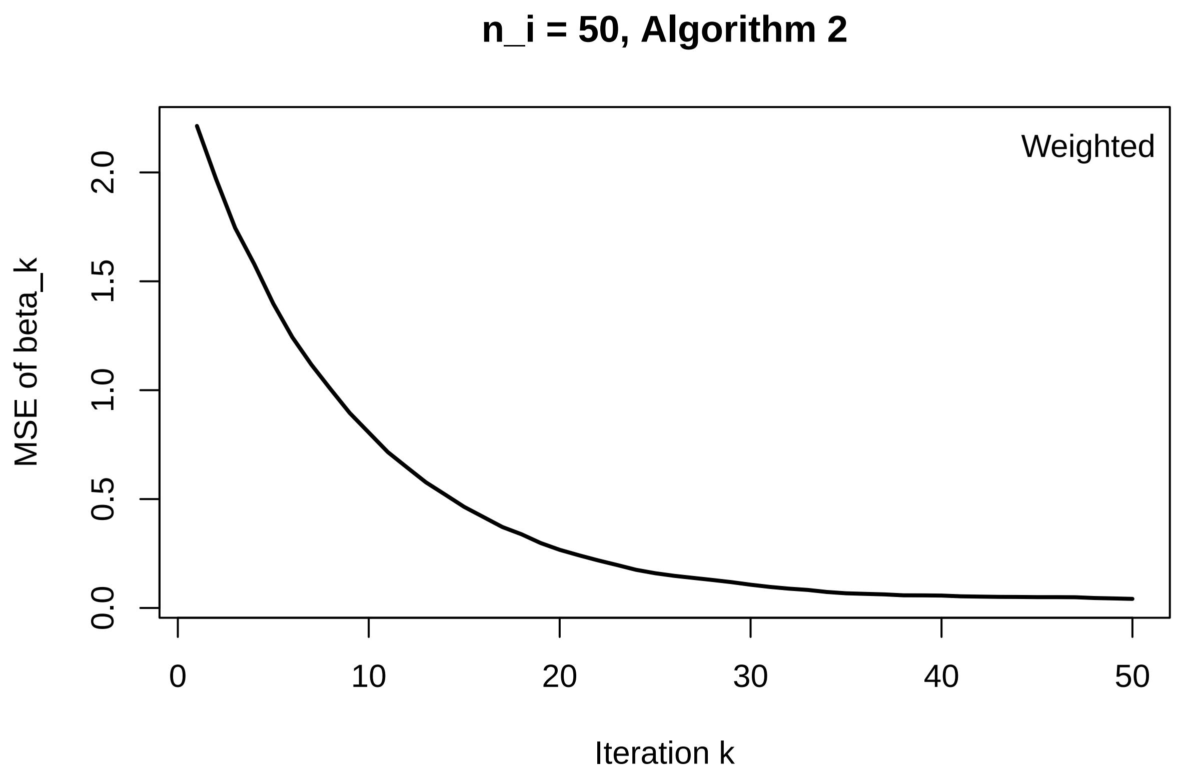
```r
all_local_beta_K1_AG2_stageI <- function(client_list,eta_1,mu,K1){

  # Find the local sample size for each client
  client_ni = rep(NA_real_, length(client_list))
  for (client_i in 1:length(client_list)){
    client_ni[client_i] = length(client_list[[client_i]]$y)
  }

  m = length(client_list)
  p = dim(client_list[[1]]$X)[2]

  # store the local beta_K1 for all clients
  local_beta_k1_results <- matrix(NA_real_, nrow = m, ncol = p)
  for (i in 1:length(client_list)){
    # Initialization beta_0 = 0
    beta_0 <- rep(0,p)

    # Define Bi
    all_local_gradient_norm_at_client_i <- rep(NA_real_,client_ni[i])
    # Calculate the norm of each local gradient
    for (j in 1:client_ni[i]){
      X_ij <- client_list[[i]]$X[j, ]
      y_ij <- client_list[[i]]$y[j]
      g_norm_ij <- gradient_rho_single_norm(X_ij, y_ij, beta_0)
      all_local_gradient_norm_at_client_i[j] <- g_norm_ij
    }
    # Let B be the 90th percentile of the norms
    Bi <- as.numeric(quantile(all_local_gradient_norm_at_client_i, probs = 0.9))

    local_beta_path_client_i <- matrix(NA_real_,nrow = K1+1, ncol = p)
    local_beta_path_client_i[1, ] <- beta_0

    # Local K1 iterations
    for (t in 1:K1){
      local_beta_path_client_i[t+1, ] <- update_local_beta_AG2_stageI(local_beta_path_client_i[t, ], eta
                                K1, client_list[[i]]$X,client_list[[i]]$y,m)
    }

    local_beta_k1_results[i, ] <- local_beta_path_client_i[K1+1, ]
  }

  return(local_beta_k1_results)
}
```

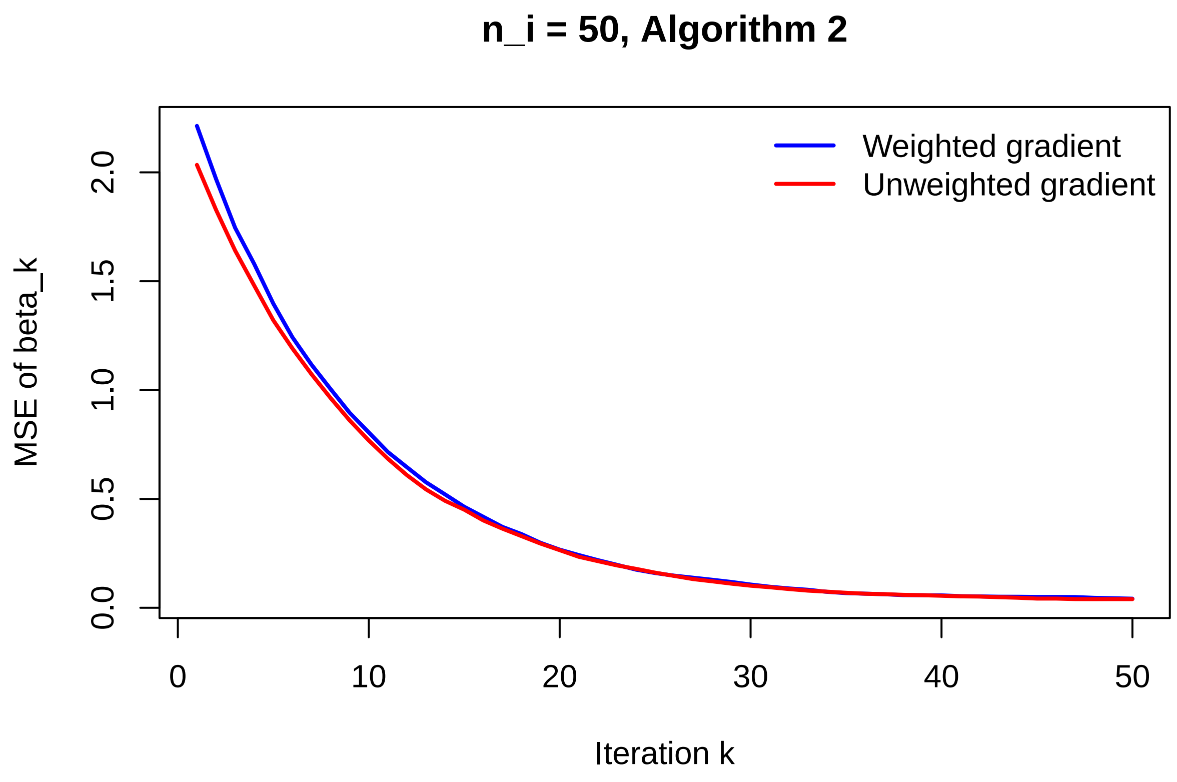
2.1 all clients have the same local sample size

```
client_ni_21 = rep(50,m)
N21 <- sum(client_ni_21)
```

```
# Store the simulation results
server_beta_K2_mse_sim_result_21 <- matrix(NA_real_, nrow = num_sim, ncol = K2)

for (sim_i in 1:num_sim){
  # Generate data and split them into m clients
  data_sim_i <- generate_data(N21, beta_true, sigma_c)
  clients_sim_i <- split_client(data_sim_i$X, data_sim_i$y, client_ni_21)

  # Stage I: K1 local iterations on each client, applying private gradient descent.
  local_beta_K1_results <- all_local_beta_K1_AG2_stageI(clients_sim_i,eta_1,mu,K1)

  # Stage II: Server takes the averages and set the private initialization
  beta_0_stageII <- colMeans(local_beta_K1_results)

  # Stage II: Run Algorithm 1 for K2 iterations with the private initialization
  beta_path_stageII <- AG1_in_AG2(clients_sim_i, client_ni_21, K2,
                                  eta_2, mu, beta_0_stageII)

  # MSE path for beta_k in the server
  mse_path <- rep(NA_real_, K2)
  for (k in 1:K2) {
    b_k <- beta_path_stageII[k + 1, ]
    mse_path[k] <- mse_beta(b_k, beta_true)
  }
  server_beta_K2_mse_sim_result_21[sim_i, ] <- mse_path
}
```

```
# Plot mse vs k
plot_mse_path21 <- colMeans(server_beta_K2_mse_sim_result_21)
plot(1:K2, plot_mse_path21, type = "l", lwd = 2,
     xlab = "Iteration k",
     ylab = "MSE of beta_k",
     main = "n_i = 50, Algorithm 2")
legend("topright", legend = c("Weighted"), bty = "n")
```



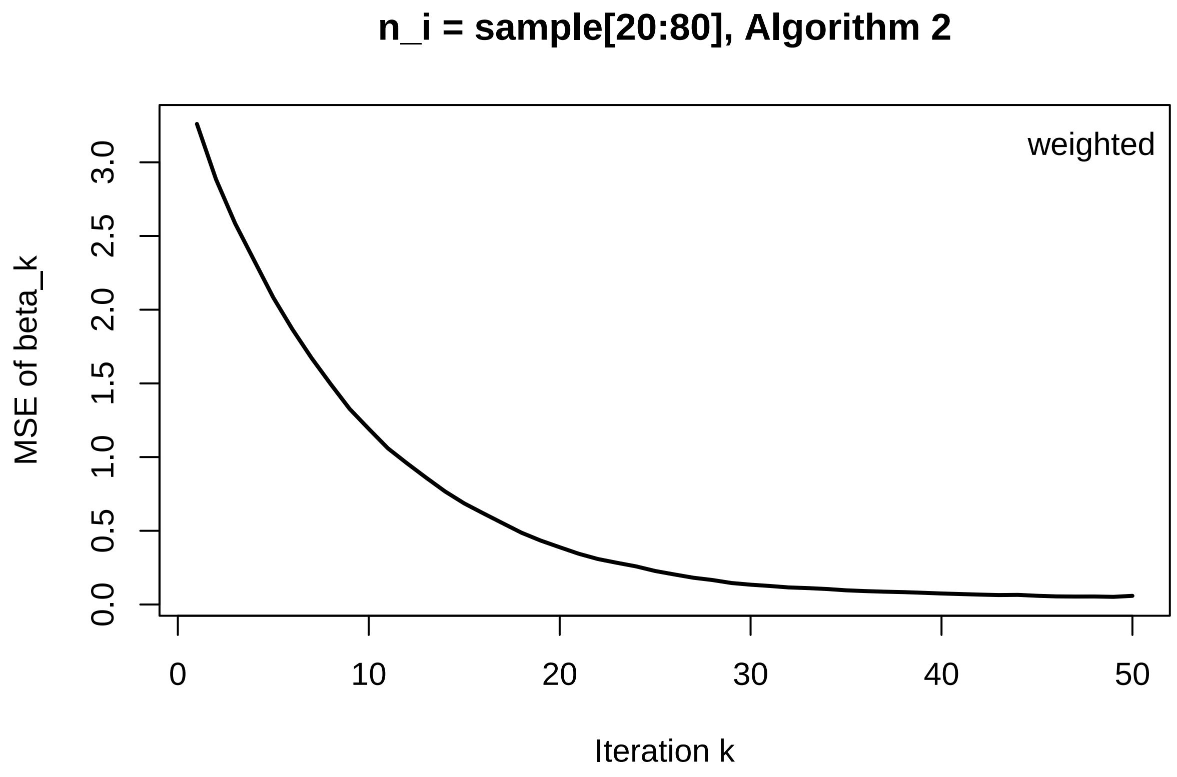


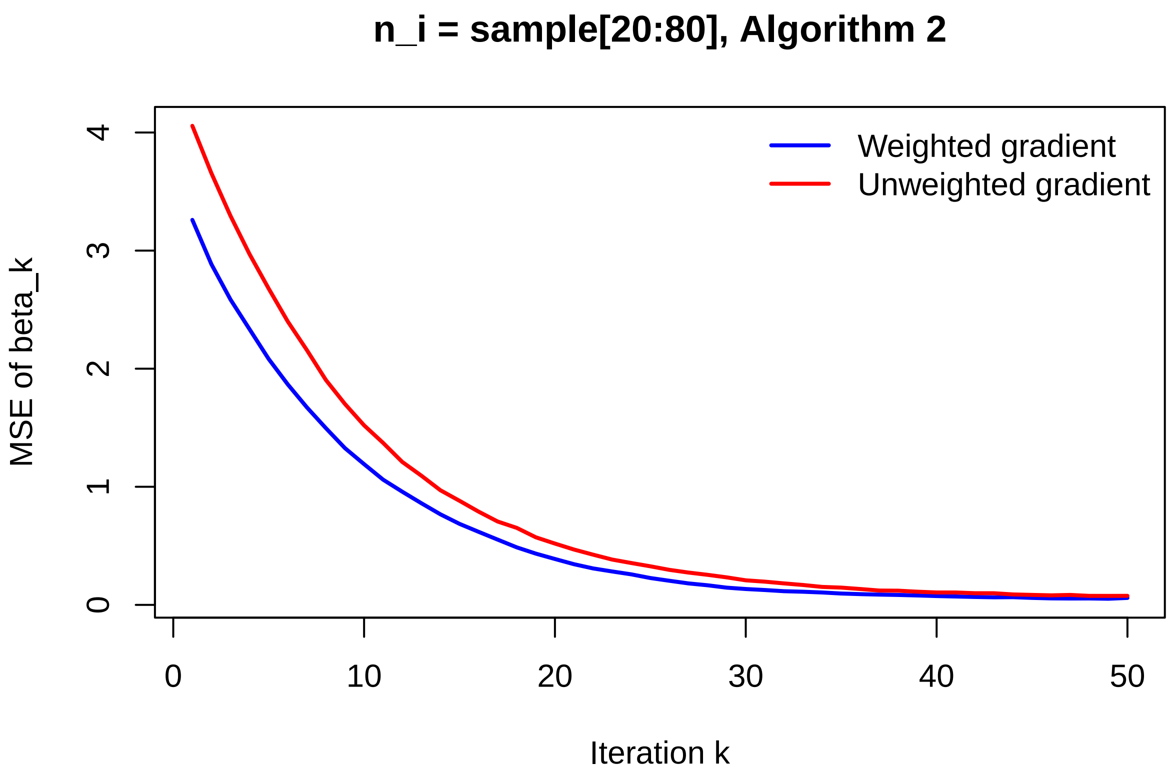
2.2 clients have different local sample size

```r
client_ni_22 = sample(20:80,m,replace = TRUE)
N22 <- sum(client_ni_22)
```

```r
# Store the simulation results
server_beta_K2_mse_sim_result_22 <- matrix(NA_real_, nrow = num_sim, ncol = K2)

for (sim_i in 1:num_sim){
  # Generate data and split them into m clients
  data_sim_i <- generate_data(N22, beta_true, sigma_c)
  clients_sim_i <- split_client(data_sim_i$X, data_sim_i$y, client_ni_22)

  # Stage I: K1 local iterations on each client, applying private gradient descent.
  local_beta_K1_results <- all_local_beta_K1_AG2_stageI(clients_sim_i,eta_1,mu,K1)

  # Stage II: Server takes the averages and set the private initialization
  beta_0_stageII <- colMeans(local_beta_K1_results)

  # Stage II: Run Algorithm 1 for K2 iterations with the private initialization
  beta_path_stageII <- AG1_in_AG2(clients_sim_i, client_ni_22, K2,
                                  eta_2, mu, beta_0_stageII)

  # MSE path for beta_k in the server
  mse_path <- rep(NA_real_, K2)
  for (k in 1:K2) {
    b_k <- beta_path_stageII[k + 1, ]
    mse_path[k] <- mse_beta(b_k, beta_true)
  }
  server_beta_K2_mse_sim_result_22[sim_i, ] <- mse_path
}
```

```
# Plot mse vs k
plot_mse_path22 <- colMeans(server_beta_K2_mse_sim_result_22)
plot(1:K2, plot_mse_path22, type = "l", lwd = 2,
     xlab = "Iteration k",
     ylab = "MSE of beta_k",
     main = "n_i = sample[20:80], Algorithm 2")
legend("topright", legend = c("weighted"), bty = "n")
```




n_i = sample[20:80], Algorithm 2
MSE of beta_k
Iteration k
Weighted gradient
Unweighted gradient
0
1
2
3
4
0
10
20
30
40
50