

SEMI-PARAMETRIC RELATIVE RISK MODEL FOR CORRELATED BINARY DATA

ANIKO SZABO

Based on the exchangeability and marginal compatibility assumptions, we propose the following semi-parametric model to describe $\lambda_k(z), k = 0, \dots, N$ as a function of covariates z :

$$\lambda_k(z) = \mu_k \times \theta(\beta' z)^k, k = 0, \dots, N, \quad (1)$$

where $\theta(\beta' z)$ is a known function with range $[0, 1]$; and μ_k is a non-parametric baseline vector of joint probabilities.

Since λ_1 equals the marginal probability of a response π , model (1) can be interpreted as a multiplicative model for it:

$$\pi(z) = \mu_1 \times \theta(\beta' z).$$

Here μ_1 is the maximal marginal probability that can be achieved by the model, since $\theta(\cdot) \leq 1$. It can also be interpreted as the probability of response for $z = z_1$ for which $\theta(\beta' z_1) = 1$ (potentially in limit).

We will allow two options: having μ_1 as a fixed pre-defined constant, or estimating it from the data.

1. INITIAL SETUP

"../R/SPreg.R" 1a≡

```
require(CorrBin)
```

◇

File defined by [1a](#), [2](#), [3b](#), [4](#), [5](#).

1.1. Exchangeable model.

$$p_{r,n} = \binom{n}{r} \sum_{j=0}^{n-r} (-1)^j \binom{n-r}{j} \lambda_{r+j} = \binom{n}{r} \sum_{s=r}^n (-1)^{(s-r)} \binom{n-r}{s-r} \lambda_s, \quad (2)$$

$$\lambda_k = \sum_{j=0}^{n-k} \frac{\binom{n-k}{j} p_{n-j,n}}{\binom{n}{n-j}} = \sum_{r=k}^n \frac{\binom{n-k}{n-r} p_{r,n}}{\binom{n}{r}}, \quad (3)$$

"../R/SPreg.R" 1b≡

```
# lambda-to-p weight matrix, rows:r, cols:s
# (-1)^(s-r)*choose(n,r)*choose(n-r,s-r)
```

```
weight_mat <- function(n){
  s <- r <- 0:n
  res <- outer(r, s, function(x,y) (-1)^(y-x)*choose(n,x)*choose(n-x,y-x))
  res
}
```

```
p_from_lambda <- function(lambda.vec, n){
  H <- weight_mat(n)
  c(H %*% lambda.vec[1:(n+1)])
}
```

```
# p-to-lambda weight matrix, rows:k, cols:r
# (choose(n-k, n-r)/choose(n,r))
```

```

weight_mat2 <- function(n){
  k <- r <- 0:n
  res <- outer(k, r, function(x,y)choose(n-x, n-y)/choose(n,y))
  res
}

lambda_from_p <- function(p.vec, n=length(p.vec)-1){
  H <- weight_mat2(n)
  c(H %*% p.vec)
}
◇
File defined by 1ab, 2, 3b, 4, 5.

```

2. DEFINING THE MODEL

Parameter estimation is based on the maximization likelihood estimator with respect to likelihood function of observed data. Under the marginal compatibility assumption, the parameters λ are independent of cluster sizes. The likelihood function based on the observed data is written as follows:

$$L = \prod_{i=1}^I f_i \log p_{r_i, n_i}(z_i), \quad (4)$$

where $p(r, n)(z)$ is the probability of observing r responses in a cluster of size n given covariates z , that can be calculated from (1) using the connection between λ s and probabilities in (2), and f_i are observation weights.

"../R/SPreg.R" 2≡

```

#' Fit semi-parametric relative risk model
#'
#'@rdname sprr
#'@param formula a one-sided formula of the form \code{cbind(r, s) ~ predictors} where \code{r} and \code{s}
#'@param data an optional matrix or data frame containing the variables in the formula \code{formula}. B
#'@param subset an optional vector specifying a subset of observations to be used.
#'@param weights an optional vector specifying observation weights.
#'@param link a link function for the binomial distribution, the inverse of which models the covaria
#'@param mu1 an optional value between 0 and 1 giving the maximal predicted marginal probability. I
#'@param start an optional list with elements named \code{beta}, \code{q}, and/or \code{mu1} giving s
#'@param control a list with parameters controlling the algorithm.
#'@return an object of class \code{sprr} with the fitted model.
#'@export
#' @importFrom stats terms model.matrix binomial dbinom glm uniroot

sprr <- function(formula, data, subset, weights, link="cloglog", mu1=NULL, start=NULL, control=list(eps=0
  fam <- binomial(link=link)
  model_fun <- fam$linkinv

  { Create model matrix from formula and data 3a }
  { Fit model 6 }

  mt <- attr(mf, "terms")
  res <- list(coefficients = beta_new, q=q_new, niter = iter, loglik=logl,
    link = link, call = mc, terms = mt,
    xlevels = .getXlevels(mt, mf),
    data_object=list(model_matrix=mm, resp=Y, n=rowSums(Y), weights=weights),
    model_fun=model_fun)
  class(res) <- "sprr"

```

```

    res
  }
  ◇
File defined by 1ab, 2, 3b, 4, 5.

```

⟨ Create model matrix from formula and data 3a ⟩ ≡

```

if (missing(formula) || (length(formula) != 3L))
  stop("'formula' missing or incorrect")
if (missing(data))
  data <- environment(formula)
mc <- match.call(expand.dots = FALSE)
m <- match(c("formula", "data", "subset", "weights"), names(mc), 0L)
m <- mc[c(1L,m)]
if (is.matrix(eval(m$data, parent.frame()))))
  m$data <- as.data.frame(data)
m[[1L]] <- quote(stats::model.frame)
mf <- eval(m, parent.frame())

# extract and check response variable
Y <- model.response(mf)
if (ncol(Y) != 2) stop("The response variable should be a 2-column matrix")

# create model matrix
mm <- model.matrix(formula, data=mf)

# extract weights
weights <- as.vector(model.weights(mf))
if (!is.null(weights) && !is.numeric(weights))
  stop("'weights' must be a numeric vector")
if (!is.null(weights) && any(weights < 0))
  stop("negative weights not allowed")
  ◇
Fragment referenced in 2.

```

Using the model (1), and the 1-to-1 relationship equation (2), for cluster size N

$$\begin{aligned}
 p_{r,N}(z) &= \binom{N}{r} \sum_{j=0}^{N-r} (-1)^j \binom{N-r}{j} \mu_{r+j} \theta(\beta'z)^{r+j} \\
 &= \sum_{y=r}^N \binom{y}{r} \theta(\beta'z)^r (1 - \theta(\beta'z))^{y-r} q_y.
 \end{aligned} \tag{5}$$

Then for other cluster sizes, using marginal compatibility,

$$p_{r,n}(z) = \sum_{s=0}^N h(r, s, n, N) p_{s,N}(z).$$

2.1. Model predictions and likelihood. We define internal functions that calculate the model-based predicted values for a set of parameters, and the log-likelihood.

"../R/SPreg.R" 3b≡

```

pred_lp <- function(beta, data_object){
  lp <- data_object$model_matrix %*% beta
  c(lp)
}

```

```

}
pred_theta <- function(beta, data_object, model_fun){
  lp <- pred_lp(beta, data_object)
  theta <- model_fun(lp)
  theta
}

pred_lambda <- function(beta, q, data_object, model_fun){
  lp <- data_object$model_matrix %*% beta
  theta <- model_fun(lp)
  N <- length(q)-1
  lambda_N <- lambda_from_p(q)
  th <- sapply(0:N, function(k)theta^k)
  lambda <- apply(th, 1, function(t)t * lambda_N)
  rownames(lambda) <- 0:N
  t(lambda)
}

pred_pvec <- function(beta, q, data_object, model_fun){
  ll <- pred_lambda(beta, q, data_object, model_fun)
  cs <- data_object$n
  pp <- lapply(seq_along(cs),
               function(i){res <- p_from_lambda(ll[i,], n=cs[i]);
                           names(res) <- 0:cs[i];
                           res})
  pp
}

pred_p <- function(beta, q, data_object, model_fun){
  pvec <- pred_pvec(beta, q, data_object, model_fun)
  rvec <- data_object$resp[,1]
  pp <- sapply(seq_along(rvec), function(i)pvec[[i]][rvec[i]+1])
  pp
}

loglik <- function(beta, q, data_object, model_fun){
  p <- pred_p(beta=beta, q=q, data_object = data_object, model_fun=model_fun)
  w <- data_object$weights
  if (is.null(w)) ll <- sum(log(p))
  else ll <- sum(ifelse(w==0, 0, w*log(p)))
  ll
}

```

File defined by [1ab](#), [2](#), [3b](#), [4](#), [5](#).

2.2. Methods for the sprr class. First, define a printing method which does not show the saved data and model matrix

"../R/SPreg.R" 4≡

```

# based on print.lm
print.sprr <- function(x, digits = max(3L, getOption("digits") - 3L),...){
  cat("\nA semi-parametric relative risk regression model fit\n")
  cat("\nCall:\n", paste(deparse(x$call), sep = "\n", collapse = "\n"),
      "\n\n", sep = "")
  if (length(coef(x))){
    cat("Coefficients:\n")
    print.default(format(x$coefficients, digits = digits),
                  print.gap = 2, quote = FALSE)
  }
  else cat("No coefficients\n")
}

```

```

if (length(x$q)){
  cat("Baseline joint probabilities (mu):\n")
  N <- length(x$q)-1
  ll <- c(lambda_from_p(x$q))
  names(ll) <- 0:N
  print.default(format(ll, digits = digits),
    print.gap = 2, quote = FALSE)
}
else cat("No baseline joint probabilities\n")

cat("Log-likelihood: ", format(x$loglik, digits=digits), "\n")
invisible(x)
}
◇

```

File defined by [1ab](#), [2](#), [3b](#), [4](#), [5](#).

The prediction method will predict for a variety of scenarios:

- Input data set:
 - the data used in the fitting;
 - new data.
- Results:
 - $p_{r,n}(z)$: the probability of observing the given r responses with cluster size n , given z ;
 - $\{p_{\cdot,n}(z)\}$: the entire vector of response probabilities for cluster size n , given z (will be a list due to varying lengths);
 - $\lambda_{1,n}(z)$: the marginal probability of response for cluster size n , given z ;
 - $\{\lambda_{\cdot,n}(z)\}$: the entire vector of joint probabilities λ for cluster size n , given z ;
 - $\beta'z$: the linear predictor value z ;
 - $\theta(\beta'z)$: the relative risk at predictor value z .

"../R/SPreg.R" 5≡

```

predict.sprr <- function(object, newdata=NULL,
  type=c("mean", "relrisk", "likelihood", "probvec", "lvec", "lp"),
  newn=NULL, ...){
  type <- match.arg(type)
  tt <- terms(object)
  if (!missing(newdata)){
    Terms <- delete.response(tt)
    m <- model.frame(Terms, newdata, xlev = object$xlevels)
    if (!is.null(cl <- attr(Terms, "dataClasses"))){
      .checkMFClasses(cl, m)
    }
    mm <- model.matrix(Terms, m)
    data_object <- list(model_matrix=mm)
    if (!missing(newn)){
      if (!(length(newn) == 1L || length(newn) == nrow(newdata)))
        stop("'newn' should have length 1 or equal to the number of rows of 'newdata'")
      data_object$n <- rep(newn, length=nrow(newdata))
    }
  } else {
    data_object <- object$data_object
  }

  if (type=="likelihood"){
    if (!missing(newdata))
      stop("Type = 'likelihood' is not available for new data. Consider using type='probvec' to get vec

```

```

    pred <- pred_p(beta=object$coefficients, q=object$q,
                  data_object=data_object,
                  model_fun=object$model_fun)
  } else
  if (type %in% c("mean", "lvec")){
    ll <- pred_lambda(beta=object$coefficients, q=object$q,
                     data_object=data_object,
                     model_fun=object$model_fun)
    pred <- if (type=="mean") ll[,2] else ll
  } else
  if (type == "lp"){
    pred <- pred_lp(beta=object$coefficients, data_object=data_object)
  } else
  if (type == "relrisk"){
    pred <- pred_theta(beta=object$coefficients, data_object=data_object,
                      model_fun=object$model_fun)
  }
  if (type == "probvec"){
    if (!missing(newdata) && missing(newn))
      stop("For prediction of probability vectors with new data, cluster sizes should be specified in 'n'")
    pred <- pred_pvec(beta=object$coefficients, q=object$q, data_object=data_object,
                     model_fun=object$model_fun)
  }
  return(pred)
}

```

File defined by [1a](#), [2](#), [3b](#), [4](#), [5](#).

3. FITTING THE MODEL VIA AN EM-MM ALGORITHM

We implement an algorithm called the Expectation Maximization Minorize-Maximize, abbreviated as EM MM, to estimate parameters. Catalina Stefanescu and Bruce W. Turnbull have proved that the marginal compatibility assumption is equivalent to assuming that clusters are from a sample of clusters sharing the same cluster size (the maximum cluster size N is a good choice and is used in our study), but some observations are completely missing at random, abbreviated as MCAR [?]. The expectation step in the EM MM algorithm can be performed based on the MCAR assumption. The estimation of the non-parametric backbone μ_k will be done through the corresponding pdf $q_t, t = 0, \dots, N$, because enforcing the complete monotonicity of μ is much more demanding than enforcing $\sum q_t = 1$. Thus the set of parameters to be estimated is $\phi = (q, \beta)$, under the restrictions $\sum_{y=0}^N q_y = 1$ and $\frac{1}{N} \sum_{y=0}^N y q_y = \mu_1$.

In the EM setup, the missing data for each cluster i comes from its representation as a MCAR sample from a cluster of size N with s_i responses. So the complete data are $\mathcal{D} = \{r_i, n_i, z_i; s_i\}_{i=1}^I$ and the observed data are $\mathcal{D} = \{r_i, n_i, z_i; s_i\}_{i=1}^I$. The complete data log-likelihood is

$$\log L_c(\phi \mid \mathcal{D}_c) = \sum_{i=1}^I f_i \log p_{s_i}(z_i) = \sum_{i=1}^I \sum_{s=0}^N f_i I(s_i = s) \log \left[\sum_{y=s}^N \binom{y}{s} \theta(\beta' z_i)^s (1 - \theta(\beta' z_i))^{y-s} q_y \right].$$

Its expectation given the observed data and a current set of parameter estimates $\phi^{(k)}$ is

$$E[\log L_c \mid \phi^{(k)}, \mathcal{D}] = \sum_{i=1}^I \sum_{s=0}^N f_i P(S_i = s \mid \phi^{(k)}, \mathcal{D}) \log \left[\sum_{y=s}^N \binom{y}{s} \theta(\beta' z_i)^s (1 - \theta(\beta' z_i))^{y-s} q_y \right].$$

$\langle \text{Fit model 6} \rangle \equiv$

$\langle \text{Define internal functions 9a} \rangle$

$\langle \text{Set initial values 9c} \rangle$

$\langle \text{Setup for E-step 8a} \rangle$

```

iter <- 0
diff <- 100
while ((diff > control$eps) & (iter < control$maxit)){
  iter <- iter + 1
  beta_old <- beta_new
  q_old <- q_new

   $\langle$  E-step 8b  $\rangle$ 
   $\langle$  M-step for beta 8c  $\rangle$ 
   $\langle$  M-step for q 9b  $\rangle$ 

  diff <- sum(abs(beta_old - beta_new)) + sum(abs(q_old - q_new))
}
logl <- loglik(beta_new, q_new, list(model_matrix=mm, resp=Y, n=rowSums(Y), weights=weights), model_f
names(beta_new) <- colnames(mm)

```

◇

Fragment referenced in 2.

3.1. **E-step.** Using the Bayes theorem,

$$e_{is}^{(k)} = P(s_i = s \mid \phi^{(k)}, \mathcal{D}) \propto h(r_i, s, n_i, N) P(S_i = s \mid \phi^{(k)}, \mathcal{D}),$$

where the coefficient of proportionality is chosen so that these values add up to 1 over $s = 0, \dots, N$, and

$$P(S_i = s \mid \phi^{(k)}, \mathcal{D}) = \sum_{y=s}^N \binom{y}{s} \theta((\beta^{(k)})' z_i)^s (1 - \theta((\beta^{(k)})' z_i))^{y-s} q_y^{(k)}.$$

3.2. **M-step.** The maximization step uses Minorize-Maximize to update the parameters ϕ . We apply Jensen's inequality to bound from below the sum within the logarithm by assigning element-wise weights to each q_y , and transforming the log sum expression to obtain an update for parameters ϕ .

$$\begin{aligned}
\log \left[\sum_{y=s}^N \binom{y}{s} \theta(\beta' z_i)^s (1 - \theta(\beta' z_i))^{y-s} q_y \right] &= \log \left[\sum_{y=s}^N w_{iys}^{(k)} \frac{\binom{y}{s} \theta(\beta' z_i)^s (1 - \theta(\beta' z_i))^{y-s} q_y}{w_{iys}^{(k)}} \right] \geq \\
&= \sum_{y=s}^N w_{iys}^{(k)} \log \left[\frac{\binom{y}{s} \theta(\beta' z_i)^s (1 - \theta(\beta' z_i))^{y-s} q_y}{w_{iys}^{(k)}} \right] = \\
&= \sum_{y=s}^N w_{iys}^{(k)} \log \left[\binom{y}{s} \theta(\beta' z_i)^s (1 - \theta(\beta' z_i))^{y-s} q_y \right] - \sum_{y=s}^N w_{iys}^{(k)} \log w_{iys}^{(k)}.
\end{aligned}$$

Therefore, the lower bound of the expected complete data log-likelihood function is:

$$\begin{aligned}
E[\log L_c(\phi) \mid \phi^{(k)}, \mathcal{D}] &\geq \\
&= \sum_{i=1}^I \sum_{s=0}^N \sum_{y=s}^N f_i e_{is}^{(k)} w_{iys}^{(k)} \log \left[\binom{y}{s} \theta(\beta' z_i)^s (1 - \theta(\beta' z_i))^{y-s} \right] + \sum_{i=1}^I \sum_{s=0}^N \sum_{y=s}^N f_i e_{is}^{(k)} w_{iys}^{(k)} \log q_y - \\
&\quad \sum_{i=1}^I \sum_{s=0}^N \sum_{y=s}^N f_i e_{is}^{(k)} w_{iys}^{(k)} \log w_{iys}^{(k)}, \quad (6)
\end{aligned}$$

where the last term does not actually depend on the parameters, and can be ignored. Also note that the terms containing β and q are separated, and can be maximized individually.

The weights at the k^{th} iteration are selected so that they add up to 1 and in the above equation equality holds for $\phi = \phi^{(k)}$,

$$w_{iys}^{(k)} = \frac{\binom{y}{s} \theta(\beta^{(k)'} z_i)^s (1 - \theta(\beta^{(k)'} z_i))^{y-s} q_y^{(k)}}{\sum_{\gamma=s}^N \binom{\gamma}{s} \theta(\beta^{(k)'} z_i)^s (1 - \theta(\beta^{(k)'} z_i))^{\gamma-s} q_\gamma^{(k)}}. \quad (7)$$

3.3. Implementation of E-step. In (6) we only need

$$e_{is}^{(k)} w_{iys}^{(k)} = \frac{f_i h(r_i, s, n_i, N) a_{iys}^{(k)}}{\sum_{t=0}^N \sum_{\gamma=t}^N h(r_i, t, n_i, N) a_{i\gamma t}^{(k)}},$$

where $a_{iys}^{(k)} = \binom{y}{s} \theta(\beta^{(k)'} z_i)^s (1 - \theta(\beta^{(k)'} z_i))^{y-s} q_y^{(k)}$ and the denominator is just a normalizing constant.

$\langle \text{Setup for E-step 8a} \rangle \equiv$

```
# replace each cluster with N(N-1)/2 clusters of size N
new <- cbind(s=rep(0:N, each=N+1), y=rep(0:N, times=N+1))
new <- new[new[, "s"] <= new[, "y"], ]

rep_idx <- rep(1:nrow(Y), each=nrow(new))
Y2 <- cbind(1:nrow(Y), Y)[rep_idx, ]
colnames(Y2) <- c("i", "resp", "nonresp")
mm2 <- mm[rep_idx, , drop=FALSE]

rep_idx2 <- rep(1:nrow(new), times=nrow(Y))
Ycomb <- cbind(Y2, new[rep_idx2, ])
```

\diamond
Fragment referenced in 6.

$\langle \text{E-step 8b} \rangle \equiv$

```
# calculate a_{iys}^k
lp <- mm2 %*% beta_old
theta <- model_fun(lp)
a <- dbinom(x=Ycomb[, "s"], size=Ycomb[, "y"], prob=theta) * q_old[Ycomb[, "y"]+1]

# calculate e_{is}^k w_{iys}^k
ew_num <- dhyper(x=Ycomb[, "resp"], m=Ycomb[, "s"], n=N - Ycomb[, "s"],
                 k=Ycomb[, "resp"] + Ycomb[, "nonresp"]) * a
ew_denom <- tapply(ew_num, list(i=Ycomb[, "i"]), sum, simplify=TRUE)
ew <- ew_num / ew_denom[Ycomb[, "i"]]
if (!is.null(weights)) ew <- weights * ew
```

\diamond
Fragment referenced in 6.

3.4. Implementation of the M-step. The first term in (6) corresponds to a weighted binomial likelihood with y as the cluster size, s as the number of successes, a link function θ^{-1} , and weights $f_i e_{is}^{(k)} w_{iys}^{(k)}$, so β can be updated using logistic regression.

$\langle \text{M-step for beta 8c} \rangle \equiv$

```
mod <- glm(cbind(Ycomb[, "s"], Ycomb[, "y"]-Ycomb[, "s"]) ~ mm2+0, family=fam, weights=ew, start=beta_ol)
beta_new <- coef(mod)
```

\diamond
Fragment referenced in 6.

The second term is a multinomial likelihood for q with the usual restriction $\sum_{y=0}^N q_y = 1$, and an additional restriction for the mean $\sum_{y=0}^N yq_y = N\mu_1$. Rewriting the second term of (6) with $c_y^{(k)} = \sum_{i=1}^I \sum_{s=0}^N f_i e_{is}^{(k)} w_{iys}^{(k)}$ and including the equality constraints via a Lagrangian, we need to maximize

$$F(q, \alpha_1, \alpha_2) = \sum_{y=0}^N c_y^{(k)} \log q_y - \alpha_1 \left(\sum_{y=0}^N q_y - 1 \right) - \alpha_2 \left(\sum_{y=0}^N yq_y - N\mu_1 \right).$$

Taking partial derivatives, and setting them to 0, we can show that the solution is

$$q_y^{(k)} = \frac{c_y^{(k)}}{1 + ry} \bigg/ \left[\sum_{x=0}^N \frac{c_x^{(k)}}{1 + rx} \right], \quad (8)$$

where $r = \hat{\alpha}_2 / \hat{\alpha}_1$ is the solution of the equation

$$f(r) = \sum_{x=0}^N \frac{c_x^{(k)}(x - N\mu_1)}{1 + rx} = 0. \quad (9)$$

To ensure $q_y \geq 0$ for all $y = 0, \dots, N$, we need $r > -\frac{1}{N}$. We reparameterize it as $r = -\frac{1}{N} + \exp(\rho)$ to enforce the constraint.

$\langle \text{Define internal functions 9a} \rangle \equiv$

```
mean_constrained_probs <- function(cc, m=NULL){
  if (is.null(m)) return(cc/sum(cc))
  N <- length(cc) - 1
  r_eq <- Vectorize(function(rho){i <- 0:N; sum(cc * (i - m)/(1+(exp(rho)-1/N)*i))})
  rho <- uniroot(r_eq, interval=c(-10, 10), extendInt="yes")
  r <- exp(rho$root) - 1/N
  q <- cc / (1 + r * (0:N))
  q <- q/sum(q)
  q
}
```

Fragment referenced in 6.

$\langle M\text{-step for } q \text{ 9b} \rangle \equiv$

```
c_vec <- tapply(ew, list(y=Ycomb[, "y"]), sum, simplify=TRUE)
if (is.null(mu1))
  q_new <- mean_constrained_probs(c_vec)
else
  q_new <- mean_constrained_probs(c_vec, N*mu1)
```

Fragment referenced in 6.

3.5. Initial values. The initial values for β can be selected using linear regression on transformed estimates of the marginal probabilities:

$$\theta^{-1} \left(\frac{\pi(z)}{\mu_1} \right) = \beta' z$$

For q , we will get the marginally compatible estimate for the pooled dataset, ensure a minimal probability of 0.01 at each value, then shift it to have mean μ_1 .

$\langle \text{Set initial values 9c} \rangle \equiv$

```
N <- max(rowSums(Y))

if (is.null(start$beta)){
  p0 <- (Y[,1] + 0.5)/(Y[,1] + Y[,2] + 1)
```

```

    if (is.null(mu1) && is.null(start$mu1))
      lp0 <- fam$linkfun(pmin(1-.Machine$double.eps, p0))
    else if (!is.null(mu1))
      lp0 <- fam$linkfun(pmin(1-.Machine$double.eps, p0/mu1))
    else if (!is.null(start$mu1))
      lp0 <- fam$linkfun(pmin(1-.Machine$double.eps, p0/start$mu1))
    if (is.null(weights))
      lm0 <- lm.fit(x=mm, y=lp0)
    else
      lm0 <- lm.wfit(x=mm, y=lp0, w=weights)
    beta_new <- coef(lm0)
  } else {
    beta_new <- start$beta
  }

  if (is.null(start$q)){
    if (is.null(weights))
      pooled <- CorrBin::CBDData(data.frame(Trt = "All", NResp = Y[,1], ClusterSize = rowSums(Y)), trt="Trt",
                                clustersize="ClusterSize", nresp="NResp")
    else
      pooled <- CorrBin::CBDData(data.frame(Trt = "All", NResp = Y[,1], ClusterSize = rowSums(Y), Freq=ceiling(Y/10)), trt="Trt",
                                clustersize="ClusterSize", nresp="NResp", freq="Freq")
    est <- CorrBin::mc.est(pooled)
    q0 <- est$Prob[est$ClusterSize == N]
    if (is.null(mu1) && is.null(start$mu1))
      q_new <- mean_constrained_probs(pmax(q0, 0.01))
    else if (!is.null(mu1))
      q_new <- mean_constrained_probs(pmax(q0, 0.01), N*mu1)
    else if (!is.null(start$mu1))
      q_new <- mean_constrained_probs(pmax(q0, 0.01), N*start$mu1)
  } else {
    q_new <- start$q
    if (!is.null(start$mu1))
      q_new <- mean_constrained_probs(q_new, N*start$mu1)
  }
}

```

◇

Fragment referenced in [6](#).