Mao's Method Implementation

2023-11-14

We consider a matrix of interest $A \in \Re^{n_1 \times n_2}$ with row covariates $X \in \Re^{n_1 \times m}$ and the model

$$A = X\beta + B$$

where B is a low rank matrix and orthogonal to X. We assume that we don't fully observe A_0 and that we have a partially observed (corrupted) version of it called Y. This follows the model

$$Y_{ij} = A_{ij} + \epsilon_{ij}$$

where ϵ_{ij} are i.i.d normal with 0 mean and finite variance.

We consider the sampling indicator $w_{ij} = 1$ if Y_{ij} is observed and 0 otherwise. w_{ij} is independent from ϵ_{ij}.

In Mao's paper, the the sampling (missingness) probabilities may depend on the variate and are modeled as

$$w_{ij} \sim Bernoulli(\theta_{ij}(x_i))$$

In the code below, and based on the loss function defined in (5) and (4), we estimate β, B using formulas (8) and (11) as written below:

$$\hat{\beta} = (X^T X + n_1 n_2 \lambda_1 I_m)^{-1} X^T (W * \hat{\theta}^* * Y) \qquad (8)$$

$$\hat{B} = \frac{1}{1 + 2(1-\alpha) n_1 n_2 \lambda_2} T_{\alpha n_1 n_2 \lambda_2}(P_{\overline{X}}(W * \hat{\theta}^* * Y)) \qquad (11)$$

Where $\lambda_1, \lambda_2, \alpha$ are the regularization hyperparameters for β and B, and

$$\hat{\theta}^* = \hat{\theta}_{ij}^{-1}(x_i) = (expit((1, x_i^T)\gamma_{ij}))^{-1} \qquad (a)$$

$$T_c(D) = U \times Diag\{(\sigma_i - c)_+\} \times V^T \qquad (b)$$

$$P_{\overline{X}} = 1 - P_X \qquad (c)$$

$$P_X = X(X^T X)^{-1} X^T \qquad (d)$$

---

1) Implementation with fixed $\lambda_1, lambda_2, alpha$

The following function estimates $\hat{\beta}$ and $\hat{B}$ as above with fixed (given) hyperparameters. I will try to use the same notations as above to avoid confusion.

```
Mao.fit <- function(Y, X, lambda.1, lambda.2, alpha){
  #' ------------------------------------------------
  #' Input: Y: partially observed A,
  #'          X: covariate matrix
  #'          lambda.1, lambda.2, alpha: hyperparameters
```

```
  #' ------------------------------------------
  #' output: list of  A, Beta_hat, B_hat
  #' ------------------------------------------
  n1 = dim(Y)[1]
  n2 = dim(Y)[2]
  m  = dim(X)[2]

  W = matrix(as.numeric(Y!=0), n1, n2)
  # The following two lines are as shown in (c) and (d)
  P_X = X %*% solve(t(X) %*% X) %*% t(X)
  P_bar_X = diag(1,n1) - P_X

  # we define the factor that will be used later:
  n1n2 = svd(t(X) %*% X)$d[1] # n1 * n2

  # The following part estimates theta (missingness probabilities)
  # using logistic regression as indicated in (a)
  theta_hat = matrix(NA, n1, n2)
  for(j in 1:n2){
    model_data = data.frame(cbind(W[,j], X))
    model_fit = glm(X1~., family=binomial(), data=model_data)
    theta_hat[, j] = 1 / predict(model_fit, type="response")
  }

  # the following is the product of W * theta_hat * Y
  W_theta_Y = Y * theta_hat # * W


  # beta hat as (8)
  beta_hat = solve(t(X) %*% X + n1n2 * lambda.1 * diag(1, m)) %*% t(X) %*% W_theta_Y

  # SVD decomposition to be used in (b)
  svdd = svd(P_bar_X %*% W_theta_Y)
  # evaluation of  (b)
  n1n2 = svdd$d[1]
  T_c_D = svdd$u %*% pmax(svdd$d - alpha*n1n2*lambda.2, 0) %*% t(sdd$v)
  # B hat as in (11)
  B_hat = (1 + 2 * (1-alpha) * n1n2 * lambda.2 )^(-1) * T_c_D
  # computing the rank of B
  rank = sum(pmax(svdd$d - n1n2 * lambda2 * alpha, 0) > 0) + m

  # Estimate the matrix as given in the model at the top
   A_hat = X %*% beta_hat + B_hat

  return(list(A_hat = A_hat, B_hat = B_hat, beta_hat = beta_hat, rank = rank)
}
```

2) Hyperparameter Optimization

```
prepare_fold_data <- function(Y, X, W) {
  n1 = dim(Y)[1]
  n2 = dim(Y)[2]
```

```
  m  = dim(X)[2]

  # The following two lines are as shown in (c) and (d)
  P_X = X %*% solve(t(X) %*% X) %*% t(X)
  P_bar_X = diag(1,n1) - P_X

  theta_hat = matrix(NA, n1, n2)
  for(j in 1:n2){
    model_data = data.frame(cbind(W[,j], X))
    model_fit = glm(X1 ~ ., family = binomial(), data = model_data)
    theta_hat[, j] = 1 / predict(model_fit, type = "response")
  }

  #---------
  # The following are partial parts of equations 8 and 11 that don't involve the hyperparameters.
  # this is useful to avoid unneccessary matrix multiplications.
  #----------
  X.X = t(X) %*% X
  # this one is for equation 8, the product n1n2 is replace with the Eigen value
  n1n2Im = svd(t(X) %*% X)$d[1]  * diag(1, m)  #n1 * n2  * diag(1, m)
  # the following is the product of W * theta_hat * Y
  W_theta_Y = Y * theta_hat
  X.W.theta.Y = t(X) %*% W_theta_Y
  svdd = svd(P_bar_X %*% W_theta_Y)
  # this one is for equation 11, the product is also replace with the Eigen value of the SVD
  n1n2 = svdd$d[1]

  return(list(X=X, X.X=X.X, n1n2Im=n1n2Im, n1n2=n1n2,
              X.W.theta.Y = X.W.theta.Y, svdd=svdd))
}

Mao.fit_optimized <- function(data, lambda.1, lambda.2, alpha){
  beta_hat = solve( data$X.X + data$n1n2Im * lambda.1) %*% data$X.W.theta.Y

  T_c_D = data$svdd$u %*% pmax(data$svdd$d - alpha*data$n1n2*lambda.2, 0) %*% t(data$sdd$v)
  # B hat as in (11)
  B_hat = (1 + 2 * (1-alpha) * data$n1n2 * lambda.2 )^(-1) * T_c_D
  # Estimate the matrix as given in the model at the top
   A_hat = data$X %*% beta_hat + B_hat

  return(A_hat)
}
```

```
Mao.cv <- function(A, X, W, nfolds=5, lambda.1_grid = seq(0,1,length=30),
                   lambda.2_grid = seq(0.9, 0.1, length=30),
                   alpha_grid = seq(0.992, 1, length=20), seed=2023){

  #' ------------------------------------------------------------------
  #' Input :
  #' A : Complete (True) A matrix as in the model above of size n1 by n2
  #' X :  Covariate matrix of size  n1 by m
  #' W : Binary matrix representing the mask. wij=1 if yij is observed. size similar to A
  #' The rest are cross validation parameters
```

```r
  #' ------------------------------------------------------------------------
  #' Output:
  #' list of best parameters and best score (minimum average MSE across folds)
  #' ------------------------------------------------------------------------

  #n1 = nrow(A)
  #n2 = ncol(A)
  #m = ncol(X)
  Y = A * W

  set.seed(seed = seed)
  indices = sample(cut(seq(1, nrow(A)), reaks=n_folds, labels=FALSE))
  best_score = Inf
  best_params = list(alpha = NA, lambda.1 = NA, lambda.2 = NA)

  fold_data = lapply(:n_folds, function(i) {
    train_indices = which(indices != i, arr.ind = TRUE)
    Y_train = Y[train_indices,]
    X_train = X[train_indices,]
    W_train = W[train_indices,]
    prepare_fold_data(Y_train, X_train, W_train)
  })

  for(alpha in alpha_grid){
    for(lambda.1 in lambda.1_grid){
      for(lambda.2 in lambda.2_grid){

        scores = numeric(n_folds)
        for(i in 1:n_folds){
          data = fold_data[[i]]
          test_indices = which(indices == i, arr.ind=TRUE)

          # compute the estimates with a modified fit function
          A_hat = Mao.fit_optimized(data, lambda.1, lambda.2, alpha)

          # Evaluate model performance using MSE
          scores[i] = mean((A[test_indices,] - A_hat)^2)
        }
        avg_score = mean(scores)

        if(avg_score < best_score){
          best_score = avg_score
          best_params = list(alpha=alpha, lambda.1=lambda.1, lambda.2=lambda.2)
        }
      }
    }
  }
  return(list(best_parameters = best_params, best_score = best_score))

}
```

Simulation Test

Below is a simulation test for the method above. I will use the same settings used in Mao's paper.

```
n1 <- n2 <- 400
m <- 20
r <- 10
X <- matrix(rnorm(n1*m), ncol = m)
beta <- matrix(rnorm(m*n2), ncol=n2)
U <- matrix(rnorm(n1*r),ncol=r)
V <- matrix(rnorm(n2*r),ncol=r)
P_X = X %*% solve(t(X) %*% X) %*% t(X)
P_bar_X = diag(1,n1) - P_X
B = P_bar_X %*% U %*% t(V)
A <- X %*% beta + B
# Computing epsilon as iid zero mean Gaussian with variance chosen such that the signal-to-noise ratio
signal_A <- sum((A - mean(A))^2) / (n1 * n2 - 1)
sigma_epsilon <- sqrt(signal_A)  # Since SNR = 1
epsilon <- matrix(rnorm(n1 * n2, mean = 0, sd = sigma_epsilon), n1, n2)
Y <- A + epsilon
missing_prob = 0.2
W <- matrix( rbinom(n1*n2, 1, (1 - missing_prob) ) , nrow = n1)



  missing_prob <- 0.7
  Z_data <- matrix(rnorm(n_cols*Z_cols), ncol = Z_cols)
  E_data <- matrix(rnorm(n_rows*n_cols), ncol = n_cols)
  A <- matrix(rnorm(n_rows*10), nrow = n_rows)
  B <- matrix(rnorm(n_cols*10), ncol = n_cols)
  C <- matrix(rnorm(n_rows*10), nrow = n_rows)
  D <- matrix(rnorm(n_cols*10), ncol = n_cols)
  Px <- X_data %*% solve(t(X_data) %*% X_data) %*% t(X_data)
  Pz <- Z_data %*% solve(t(Z_data) %*% Z_data) %*% t(Z_data)
  Pxp <- diag(dim(X_data)[1]) - X_data %*% solve(t(X_data) %*% X_data) %*% t(X_data)
  Pzp <- diag(dim(Z_data)[1]) - Z_data %*% solve(t(Z_data) %*% Z_data) %*% t(Z_data)

  X_coeff <- matrix(runif(X_cols*n_cols, -1, 1), ncol = n_cols)
  Z_coeff <- matrix(runif(Z_cols*n_rows, -1, 1), ncol = n_rows)
  D_coeff <- matrix(runif(n_rows*10, -1, 1), nrow = n_rows) %*% matrix(runif(n_cols*10, -1, 1), ncol = n

  Y_data <- X_data%*%X_coeff + t(Z_data%*%Z_coeff) + Pxp %*% D_coeff %*% Pzp + E_data

  #*************
  output <- Y_data
  row_input <- X_data
  col_input <- Z_data
  mask <- W_data
cv_ratio_SMC_x = SMCfit_cv(Y_data, X_data, mask, nfolds = 5,
                           tau1_grid = seq(0, 2, length = 20), tau2_grid = seq(0.9, 0.1, length = 20),
                           alpha_grid = seq(0.992,1, length = 10))  # Choose the tuning parameter by cros

fit_mao_x <- SMCfit(Y_data*mask, X_data, cv_ratio_SMC_x$tau_beta_ratio, cv_ratio_SMC_x$tau_svd_ratio, cv
```

```r
Y_pred_x <- fit_mao_x$Ahat

test_indices <- which(W_data==0, arr.ind = TRUE)
mao_error_x <- Y_data - Y_pred_x
mao_error_x <- mao_error_x[test_indices]
mao_error_x <- sqrt(mean(mao_error_x**2))
mao_error_x
```