An applicaton of Bayesian variable selection for finite mixture model of linear regressions

Francesco Fabbri

22 marzo 2018

1. Introduction

2. Data

3. Model

3.1 Priors

Looking at the priors of the model, we can define the initialized parameters of our Gibbs-sampling.

3.1.1 Mixture proportions.

First, we define the vector ρ_0 of the mixture proportions: it's built sampling the values from a conjugate Dirichlet prior distribution:

$$\rho \sim Dirichlet(\alpha_1, ..., \alpha_M)$$

Where M is the number of the possible sub-populations. In the case of $M = 2$, we choose a not-informative value equal to 2 for each α.

```
initialize_rho = function(k, n_subpop){
  arr = c()
  for(x in 1:n_subpop){
    arr = c(arr, n_subpop)
    }
  vec = rdirichlet(k,arr)
  return(vec)
}
```

3.1.2 Latent indicator variable r.

In each mixture component of the regression model, the prior distributions of the indicator variables r_{mj} are assumed to be independent $Bernoulli(d_{mj})$ for $j = 1, ..., p$. So, looking at the joint distribution of r_m:

$$\pi(r_m) = \prod_{j=1}^{p} d_{mj}(1 - d_{mj})^{1-r_{mj}}$$

We can sample the initial value of each r_{mj} from a non-informative $Bernoulli(0.5)$.

```
initialize_r = function(x_matrix, n_subpop){
  # r - active variables
  vec_bern = c()
  for(x in 1:n_subpop){
    vec_bern = rbind(vec_bern,
                rbinom(
                  dim(x_matrix)[2],
```

```
                    size  = 1,
                    prob = 1))
  }
  vector_r_0 = vec_bern
  colnames(vector_r_0) = colnames(x_matrix)
  return(vector_r_0)
}
```

3.1.3 Latent indicator variable z.

Each z_i is generated from a multinomial distribution, i.e. $z_i \sim Multinomial(\rho_1, .., \rho_M)$

```
# initialize z
initialize_z = function(x_matrix, vector_rho_0){

  sample_z = rmultinom(dim(x_matrix)[1],
                       1,
                       prob = vector_rho_0)

  sample_z = t(sample_z)
  z_array = sample_z

  #vector_z = c(rep(0,60), rep(1, 140))
  #new_z= vector_z
  #z_array[which(new_z == 1),1] = 1
  #z_array[which(new_z != 1),1] = 0
  #z_array[which(z_array[,1] == 1),2] = 0
  #z_array[which(z_array[,1] != 1),2] = 1

  return(z_array)
}
```

3.1.4 Priors definition for β and σ_m

The prior of each $\beta_m(r_m)$ is assumed to be the following g-prior:

$$\beta_m(r_m) \sim N\left(\hat{\beta}_m^{\lambda_m}(r_m), g_m \sigma_m^2 [X'_m(r_m) X_m(r_m)]^{-1}\right)$$

where:

- $\hat{\beta}_m^{\lambda_m}(r_m) = w_m(r_m) X'_m(r_m) Y_m$
- g_m is a positive arbitrary number to fix. It's usually equal to n_m, which is the size of the sub-population m.
- $w_m(r_m) = [X'_m(r_m) X_m(r_m) + \lambda_m I]^{-1}$
- λ_m is the ridge parameter: it's used when we cannot derive the inverse of $X'_m(r_m) X_m(r_m)$. With $\lambda_m = 0$ we have $w_m(r_m) = [X'_m(r_m) X_m(r_m)]^{-1}$
- $\sigma_m^2 \sim IG\left(\frac{a_{m_0}}{2}, \frac{b_{m_0}}{2}\right)$. We could assign to a_{m_0} and b_{m_0} some non-informative values (i.e. both equals to 0.01) or we could even fix it equals to 1, in order to avoid uncontrolled values sampled of σ_m.

```
# Beta, Beta-hat, sigma and w_m_inv
initialize_theta = function(n_subpop,
                            vector_z_0,
                            vector_r_0,
                            vector_rho_0,
```

```
                            x_matrix){
# initialize w_m
w_m_inv_0 = list()
for(idx_m in 1:n_subpop){
  # define X_m(r_m)
  x_m_rm_0 = as.matrix(x_matrix[which(vector_z_0[,idx_m] == 1),
                                which(vector_r_0[idx_m,] == 1),
                                drop=FALSE])

  # check if the matrx X_m(r_m)'*X_m(r_m) is singular or not
  if(is.singular.matrix(t(x_m_rm_0)%*%x_m_rm_0)){
    # w_m(r_m)^(-1) = (X_m(r_m)'*X_m(r_m) + lambda_m*I)
    # lambda_m = 1/p (p is the number of covariates)
    one_w_m_inv = t(x_m_rm_0)%*%x_m_rm_0 + (1/dim(x_m_rm_0)[2])*diag(dim(x_m_rm_0)[2])
    }
  else{
    # w_m(r_m)^(-1) = (X_m(r_m)'*X_m(r_m))
    one_w_m_inv = t(x_m_rm_0)%*%x_m_rm_0
    }
  # add inverse of w_m(r_m) to vector
  w_m_inv_0[[idx_m]] = one_w_m_inv
}

# initialize beta_hat, beta and sigma
vector_beta_hat_0 = list()
vector_beta_0 = list()
vector_sigma_0 = list()

for(idx_m in 1:n_subpop){
  # beta_hat = [X_m(r_m)'*X_m(r_m)]^(-1) * X_m(r_m)'*Y_m
  x_m_rm_0 = as.matrix(x_matrix[which(vector_z_0[,idx_m] == 1),
                                which(vector_r_0[idx_m,] == 1),
                                drop=FALSE])
  w_m = solve(w_m_inv_0[[idx_m]])
  selected_y = y[which(vector_z_0[,idx_m] == 1), drop=F]
  beta_hat_0 = w_m%*%t(x_m_rm_0)%*%selected_y

  # now build the variance of beta, beta_variance = g_m * sigma_m^2 * w_m(r_m)
  # g_m
  g_m = sum(vector_z_0[,idx_m])
  # sigma_0 - selected value equals to 1 in order to avoid
  sigma_0 = 1 #rinvgamma(1, 0.1, 0.1)

  # beta_variance
  var_beta_m_0 = g_m*sigma_0*solve(w_m_inv_0[[idx_m]])

  # sample of beta_m(r_m)
  beta_m_0 = mvrnorm(n = 1, mu = beta_hat_0, Sigma = var_beta_m_0)

  # update arrays of beta, beta_hat and sigma
  vector_beta_hat_0[[idx_m]] = as.matrix(beta_hat_0)
  vector_beta_0[[idx_m]] = as.matrix(beta_m_0)
  vector_sigma_0[[idx_m]] = sigma_0
```

```
  }
  # theta initialization
  theta = list('beta' = vector_beta_0,
               'beta_hat' = vector_beta_hat_0,
               'sigma' = vector_sigma_0,
               'rho' = vector_rho_0,
               'r' = vector_r_0,
               'old_beta' = vector_beta_0,
               'w_m_inv' = w_m_inv_0)
}
```

Finally, we can define the main function, used to initialize the parameters useful for running the algorithm below.

```
Initialization = function(x_matrix, y, n_subpop){
  # rho_0
  vector_rho_0 = initialize_rho(1, n_subpop)
  # r - active variables
  vector_r_0 = initialize_r(x_matrix,n_subpop)
  # z - sub-groups
  vector_z_0 = initialize_z(x_matrix,vector_rho_0)

  # theta
  theta = initialize_theta(n_subpop,
                           vector_z_0,
                           vector_r_0,
                           vector_rho_0,
                           x_matrix)

  return(list('theta' = theta, 'vector_z' = vector_z_0))
}
```

4.1 Posterior

Given the following likelihood, combined with the described priors:

$$l(y, z|\theta) = \prod_{i=1}^{n} \rho_{z_i} f(y_i|\theta_{z_i}) = \prod_{i=1}^{M} \rho_m^{n_m} \left[\prod_{i \in G_m} f(y_i|\theta_{z_i}) \right]$$

We can compose the posterior given by:

$$p(\theta, z|y) \propto \prod_{i=1}^{M} \rho_m^{n_m} \left[\prod_{i \in G_m} f(y_i|\theta_{z_i}) \right] \pi(\theta)$$

5.1 Gibbs Sampling

Given the posterior obtained by the combination of the priors and the likelihood we provide the **full conditional distributions** from which the algorithm samples the posterior parameters.

- **1. Update latent variable Z**. We update the values of the latent variable z which determines the structure of each group. To do this, for each observation we sample the z value form the following conditional distribution:

$$P(z_i = m|\theta, y) = \frac{\rho_m f(y_i|\theta_m)}{\sum_{m=1}^{M} \rho_m f(y_i|\theta_m)}$$

In other words, for each sub-population we compute the probability to assign a specific observation to it, then we sample the value of z from each distribution. The density function at the numerator corresponds to the normal density related to y, where:

$$[y_i|\theta_m] \sim N\Big(x_i(r_m)\beta_m(r_m), \sigma_m^2\Big)$$

Where $\theta_m = (\beta_m, \sigma_m^2, \rho_m, r_m)$.

```
# Update Z (GetPosterior, UpdateArrayZ)  --------------------------------
GetPosteriorZ = function(obs, theta_array, n_subpop){
  num = c()
  for(idx_m in 1:n_subpop){
    selected_beta = theta_array[['beta']][[idx_m]]
    label_beta = rownames(selected_beta)
    one_x = obs[label_beta]
    mu_normal = one_x[label_beta]%*%selected_beta
    sigma_m = theta_array[['sigma']][[idx_m]]
    f_norm = dnorm(x = obs[1], mean =  mu_normal, sd = sqrt(sigma_m))
    rho_m = theta_array[['rho']][[idx_m]]
    prod_f_rho = rho_m*f_norm
    num = c(num, prod_f_rho)
  }
  prob_z = num/sum(num)
  return(sample(x = c(1,0), size=1, prob = prob_z))
}



UpdateArrayZ = function(theta_array, dataset){
  vector_z = c()
  for(idx_obs in 1:dim(dataset)[1]){
    obs = dataset[idx_obs,]
    vector_z = c(vector_z, GetPosteriorZ(obs, theta_array, n_subpop))
  }
  #vector_z = c(rep(0,60), rep(1, 140))
  return(vector_z)
}
```

- **2. Update ρ.** The conditional distribution of ρ, given z, is the following:

$$\rho \sim Dirichlet(n_1 + \alpha_1, ..., n_M + \alpha_M)$$

Where $n_1, n_2, ..., n_M$ are the cardinality of each sub-population.

```
UpdateRho = function(z_array){
  vector_n_proportion = colSums(z_array) + 2
  vector_rho = rdirichlet(1,vector_n_proportion)
  return(vector_rho)
}
```

- **3. Update sub-populations.** At each iteration we update the sub-populations, looking at the two latent variables z and r.

```
### IMP: update subpopulation by z and r
UpdateSubPop = function(theta_array, z_array, x_matrix, n_subpop){
  new_x = list()
  for(idx_m in 1:n_subpop){
    one_for_subpop = x_matrix[which(z_array[,idx_m] == 1),
                              which(theta_array[['r']][idx_m,] == 1),
                              drop=FALSE]
    new_x[[idx_m]] = one_for_subpop
  }
  return(new_x)
}
```

- **4. Update $w_m(r_m)$.** As for the previous step, we need to update the matricies involved in our algoritm: in this case, we update the $p \times p$ matrix $w_m(r_m)$. At each iteration we check whether the updated matrix $X'_m(r_m)X_m(r_m)$ is singular or not: in the first case, the ridge parameter λ_m is added to the computed matrix otherwise not. This parameter is equal to $1/p$, where p is the number of covariates used in the specific sub-population. Adding λ_m we have $w_m(r_m) = \left(X'_m(r_m)X_m(r_m) + \lambda_m I\right)^{-1}$

```
# check if the matrix X_m_rm'X_m_rm is singular
UpdateW_inv = function(theta_array, covariates, n_subpop){
  for(idx_m in 1:n_subpop){
    x_product = t(covariates[[idx_m]])%*%covariates[[idx_m]]
    if(is.singular.matrix(x_product)){
      lambda_m = (1/dim(covariates[[idx_m]])[2])
      identity_matrix = diag(dim(covariates[[idx_m]])[2])
      one_w_m_inv = x_product + lambda_m*identity_matrix
      }
    else{
      one_w_m_inv = x_product
      }
    theta_array$w_m_inv[[idx_m]] = one_w_m_inv
  }
  return(theta_array)
}
```

- **5. Update σ_m and $\hat{\beta}_m(r_m)$.** In order to sample σ^2_m we need first to update the parameter $\hat{\beta}^{\lambda_m}_m(r_m)$, where:

$$\hat{\beta}^{\lambda_m}_m(r_m) = w_m(r_m)X'_m(r_m)Y_m$$

  So now, we need to sample the variance of each subpoulation, updating first the parameters of the full conditional, which is an inverse gamma:

$$\sigma^2_m \sim IG\left(\frac{a_m}{2}, \frac{b_m}{2}\right)$$

  We have:

- $a_m = n_m + q_m + a_{m_0}$, where $q_m = \sum_i^p r_{mj}$, i.e. the number of active covariariates in the sub-population
- $b_m = \left[Y_m - X_m(r_m)\beta_m(r_m)\right]'\left[Y_m - X_m(r_m)\beta_m(r_m)\right] + \frac{\left[\beta_m(r_m) - \beta^{\lambda_m}_m(r_m)\right]' w^{-1}_m(r_m)\left[\beta_m(r_m) - \beta^{\lambda_m}_m(r_m)\right]}{n_m} + b_{m_0}$

In this case, the sampled parameters depends on z, r_m, β_m

```
# Update sigma and BetaHat ----

# function for updating Beta Hat
UpdateBetaHat = function(theta_array, z_array, covariates){
```

```
  vector_beta_hat = list()
  for(idx_m in 1:length(theta_array$beta)){
    w_m = solve(theta_array$w_m_inv[[idx_m]])
    y_m = y[which(z_array[,idx_m] == 1)]
    beta_hat = w_m%*%t(covariates[[idx_m]])%*%y_m
    vector_beta_hat[[idx_m]] = beta_hat
  }
  return(vector_beta_hat)
}

### update sigma (a and b)
UpdateSigma = function(theta_array,z_array,covariates){

  # update a_m, b_m and sigma_m
  vector_q = rowSums(theta_array[['r']])
  vector_n_proportion = colSums(z_array)

  ### update a ###
  vector_a = vector_n_proportion + vector_q + 0.001

  # update beta-hat
  theta_array[['beta_hat']] = UpdateBetaHat(theta_array, z_array, covariates)

  # update_b
  vector_b = c()
  for(idx_m in 1:length(theta_array$beta)){
    selected_beta = theta_array[['beta']][[idx_m]]
    x_m_beta_m = covariates[[idx_m]]%*%selected_beta
    num_b = y[which(z_array[,idx_m] == 1)] - x_m_beta_m
    num_b = t(num_b)%*%num_b
    selected_beta_hat = theta_array[['beta_hat']][[idx_m]]
    diff_beta_beta_hat = selected_beta - selected_beta_hat
    w_m_inv = theta_array$w_m_inv[[idx_m]]
    frac_num = t(diff_beta_beta_hat)%*%w_m_inv%*%diff_beta_beta_hat
    second_term = frac_num/vector_n_proportion[[idx_m]]
    one_b = num_b + second_term + 0.001
    vector_b = c(vector_b, one_b)
  }

  # sample sigma_m
  vector_sigma = c()
  for(idx_m in 1:length(theta_array$beta)){
    sigma_sample = rinvgamma(n = 1, shape = vector_a[idx_m]/2, scale = vector_b[idx_m]/2)
    vector_sigma = c(vector_sigma, sigma_sample)
  }
  theta_array$sigma = vector_sigma
  return(theta_array)
}
```

- **6. Update β.** The conditional distributin of β_m is given by a multivariate normal distribution (MVN). Indeed, we have:

$$\beta_m(r_m) \sim MVN\Big(\mu_m, \Omega_m\Big)$$

  Where:

- $\mu_m = \Omega_m\left(\frac{n_m X_m(r_m) Y'_m + w_m^{(-1)}(r_m)\hat{\beta}_m^{\lambda_m}(r_m)}{n_m \sigma_m^2}\right)$ ($p \times 1$ vector)
- $\Omega_m^{-1} = \left[\frac{n_m X'_m(r_m) X_m(r_m) + w_m^{-1}(r_m)}{n_m \sigma_m^2}\right]$ ($p \times p$ matrix)

```
# Update Beta ----
UpdateBeta = function(z_array,covariates,theta_array){
  # initialization: Omega, n_m, w_m
  vector_omega = list()

  # update beta
  for(idx_m in 1:length(theta_array$beta)){
    proportion = sum(z_array[,idx_m])
    cov_multiplied = t(covariates[[idx_m]])%*%covariates[[idx_m]]
    w_m_inv = theta_array$w_m_inv[[idx_m]]
    num_omega = proportion*cov_multiplied + w_m_inv
    sigma_m = theta_array$sigma[[idx_m]]
    one_inverse_omega = num_omega/(proportion*sigma_m)
    omega = solve(one_inverse_omega)
    vector_omega[[idx_m]] = omega
  }
  theta_array$omega = vector_omega

  # mu
  vector_mu = list()
  for(idx_m in 1:length(theta_array$beta)){
    proportion = sum(z_array[,idx_m])
    x_y_multiplied = t(covariates[[idx_m]])%*%y[which(z_array[,idx_m] == 1)]
    first_term = proportion*x_y_multiplied
    w_m_inv = theta_array$w_m_inv[[idx_m]]
    beta_hat = theta_array$beta_hat[[idx_m]]
    second_term = w_m_inv%*%beta_hat
    one_omega = vector_omega[[idx_m]]
    sigma_m = theta_array$sigma[[idx_m]]
    mu = one_omega%*%(first_term + second_term)
    mu = mu/(proportion*sigma_m)
    vector_mu[[idx_m]] = mu
  }

  # sample of beta
  vector_beta = list()
  for(idx_m in 1:length(theta_array$beta)){
    estimate = as.matrix(mvrnorm(n=1,
                                 mu =vector_mu[[idx_m]],
                                 Sigma = vector_omega[[idx_m]]))
    vector_beta[[idx_m]] = estimate
    for(cov_name in rownames(estimate)){
      theta_array$old_beta[[idx_m]][cov_name,1] = estimate[cov_name,1]
    }
  }
  #print('Vector beta')
  #print(vector_beta)
  theta_array$beta = vector_beta
  return(theta_array)
}
```

- **7 Update r.** In the case of the 2nd latent variable, which addressess the task to select the right covariates for each subgroup, we have the following probability distribution:

$$p(r_{mj} = 1|\theta_m) = \frac{1}{1 + \exp\{l_n(\theta_m|r_{mj} = 0) - l_n(\theta_m|r_{mj} = 1)\}}$$

And, obviously:

$$p(r_{mj} = 0|\theta_m) = 1 - p(r_{mj} = 1|\theta_m)$$

Now, $l_n(\theta_m|.)$ is the log-likelihood of our model and there are two possible scenarios that affect the values of this function. Looking at a single r_{mj}, if at the time of the computation of the probability the covariate j in the sub-group m is active, then we can build easily the 2 log-likelihood. The first ($l_n(\theta_m|1)$) would be computed just using all the active covariates, while for the second one ($l_n(\theta_m|0)$) we just need to remove the β_j from the computation. In details:

- $l_n(\theta_m|r_{mj} = 1) = \sum_{i \in G_m} f\left(y_i \middle| x_i(r_m)\beta_m(r_m), \sigma_m^2\right)$
- $l_n(\theta_m|r_{mj} = 0) = \sum_{i \in G_m} f\left(y_i \middle| x_i(r_{m(-j)})\beta_{m(-j)}(r_{m(-j)}), \sigma_m^2\right)$

Where $m(-j)$ imposes to take all the covariates but j considered in the sub-population m.

Instead, when a covariate j is already deactivated, we need to compute $l_n(\theta_m|r_{mj} = 1)$ using an "older value" of the β_{mj}. In other words, we use the last active beta computed during the run. We have:

$$l_n(\theta_m|r_{mj} = 1) = \sum_{i \in G_m} f\left(y_i|x_i(r_m)\beta_m^*(r_m), \sigma_m^2\right)$$

Where $\beta_m^*(r_m) = [\beta_1, .., \beta_j^*.., \beta_p]$ with β_j^* which is the last non-zero value of the coefficient j in the subgroup m computed by the algorithm.

```
# Update r (and LogLike) ----

### Generate LogLike
ComputeProb = function(theta_array,vector_z, idx_m,single_label){
  # compute the prob of having active a specific variable in a sub-population
  active_covariates = labels(which(theta_array$r[idx_m,] == 1))
  if(single_label %in% active_covariates){

    ### 1st case: we can compute easily l_n(0) and l_n(1)

    # create loglike with 0 and 1
    mu_vector = c()
    for(idx_obs in which(vector_z[,idx_m] == 1)){
      specific_covariates = x_matrix[idx_obs,active_covariates ,drop = F]
      beta_m = theta_array$beta[[idx_m]]
      mu = specific_covariates%*%beta_m
      mu_vector = c(mu_vector, mu)
    }
    one_y = y[which(vector_z[,idx_m] == 1)]
    sigma_m = theta_array$sigma[[idx_m]]
    prob_1 = dnorm(x = one_y,
                   mean = mu_vector,
                   sd=sqrt(sigma_m))
    prob_1 = sum(log(prob_1))
```

```
  # prob0
  without_label = active_covariates[active_covariates!= single_label]
  mu_vector = c()
  for(idx_obs in which(vector_z[,idx_m] == 1)){
    specific_covariates = x_matrix[idx_obs,without_label ,drop = F]
    beta_m_without_j = theta_array$beta[[idx_m]][without_label,]
    mu = specific_covariates%*%beta_m_without_j
    mu_vector = c(mu_vector, mu)
  }
  one_y = y[which(vector_z[,idx_m] == 1)]
  sigma_m = theta_array$sigma[[idx_m]]
  prob_0 = dnorm(x = one_y,
                 mean = mu_vector,
                 sd=sqrt(sigma_m))
  prob_0 = sum(log(prob_0))
  probab = c(prob_0, prob_1)
  res = probab
}
else{

  ### 2nd case: we need to re-activate the variable,
  ### using the last non-zero value computed before

  #prob0
  mu_vector = c()
  for(idx_obs in which(vector_z[,idx_m] == 1)){
    specific_covariates = x_matrix[idx_obs,active_covariates ,drop = F]
    beta_m = theta_array$beta[[idx_m]]
    mu = specific_covariates%*%beta_m
    mu_vector = c(mu_vector, mu)
  }
  one_y = y[which(vector_z[,idx_m] == 1)]
  sigma_m = theta_array$sigma[[idx_m]]
  prob_0 = dnorm(x = one_y,
                 mean = mu_vector,
                 sd=sqrt(sigma_m))
  prob_0 = sum(log(prob_0))

  # prob1
  mu_vector = c()
  # add the deactivated covariate j, taking its last computed value (beta*)
  beta_m = theta_array$beta[[idx_m]]
  last_value_computed = as.matrix(theta_array$old_beta[[idx_m]][single_label,])
  reactivated = rbind(beta_m, last_value_computed)
  # order by names
  reactivated = as.matrix(reactivated[order(rownames(reactivated)),])
  labels_reactivated = rownames(reactivated)
  for(idx_obs in which(vector_z[,idx_m] == 1)){
    mu = x_matrix[idx_obs,labels_reactivated,drop = F]%*%reactivated
    mu_vector = c(mu_vector, mu)
  }
  one_y = y[which(vector_z[,idx_m] == 1)]
  sigma_m = theta_array$sigma[[idx_m]]
```

```
    prob_1 = dnorm(x = one_y,
                   mean = mu_vector,
                   sd=sqrt(sigma_m))
    prob_1 = sum(log(prob_1))

    # define the probabilities
    probab = c(prob_0, prob_1)
    res = probab
  }
  return(res)
}

# Update r ---
UpdateR = function(theta_array, vector_z){
  # initialize r
  old_r_array = theta_array$r
  new_r_array = theta_array$r
  list_likelihood = list()
  n_subpop = dim(theta_array$r)[1]
  for(idx_m in 1:n_subpop){
    # append the likelihood in a matrix
    list_likelihood[[idx_m]] = matrix(0,
                                      nrow = dim(theta_array$r)[2],
                                      ncol = dim(vector_z)[2])
    rownames(list_likelihood[[idx_m]]) = colnames(theta_array$r)

    # compute probabilities and sample r
    for(single_label in colnames(theta_array$r)){
      if(single_label!='intercept'){
        likelihood = ComputeProb(theta_array,vector_z, idx_m,single_label)
        list_likelihood[[idx_m]][single_label,] = likelihood
        one_prob = 1/(1 + exp(likelihood[1] - likelihood[2]))
        probability = c(1- one_prob, one_prob)
        one_r = sample(x = 0:1, size = 1, prob = probability)
        new_r_array[idx_m,single_label] = one_r
        # update label (keep (1) or discard (0))
      }
    }


    ### at this point we need to: activate/deactive the covariates (changing beta values)

    # reactivate beta by r
    new_active = c()
    for(single_label in colnames(theta_array$r)){
      if((old_r_array[idx_m,single_label] == 0) & (new_r_array[idx_m,single_label] == 1)){
        new_active = c(new_active, single_label)
        beta_m = theta_array$beta[[idx_m]]
        last_value = as.matrix(theta_array$old_beta[[idx_m]][single_label,] )
        reactivated = rbind(beta_m, last_value)
        reactivated = as.matrix(reactivated[order(rownames(reactivated)),])
        theta_array$beta[[idx_m]] = reactivated
      }
```

```
    }
  }

  # deactivate the beta by r
  theta_array$r = new_r_array
  for(idx_m in 1:length(theta_array$beta)){
    active_labels = names(which(theta_array$r[idx_m,]==1))
    theta_array$beta[[idx_m]] = as.matrix(theta_array$beta[[idx_m]][active_labels,])
  }
  theta_array$probalities = list_likelihood
  return(theta_array)
}
```

5.2 Variable selection for the 2 latent variables

After a sufficient number of samples of parameters are drawn from the posterior distribution by the Gibbs sampler, they are used for posterior inference.

Laten variable r. In order to determine the active variables of the linear regression model of each subpopulation, we collect the posterior samples of r_{mj}'s and adopt the **median probability criterion**. So, we decide to activate or not a variable j in a specific subpopulation m computing the following probability:

$$p(r_{mj}|y) = \frac{1}{K}\sum_{k=1}^{K} I\{r_{mj}^{(k)}\} \geq \frac{1}{2}$$

Latent variable z. Looking at the variable z, we assign an observation y_i to the mth sub-population if:

$$p(z_i = m|y) = \max_{g \in 1,..M} \sum_{k=1}^{K} I\{z_i^{(k)} = g\}$$

6. Simulation

6.1 Setting

We choose to simulate our model generating 200 observations from a mixture model of two linear regressions (M = 2), given by:

$$y \sim \rho N(x'\beta_1, 1) + (1-\rho)N(x'\beta_2, 1)$$

Each observation of the independent variable x is generated by a 5-dimensional multivariate normal distribution with mean vector 0 and covariance matrix $\sum$. Let $\sum(i,j)$ denote the (i,j) entry of $\sum$, where $\sum(i,j) = \pi^{|ij|}$ where $\pi = 0.5$. Then, the regression coefficients are set to be $\beta_1 = (1,0,0,3,0)$ and $\beta_2 = (1,2,0,0,3)$.

```
### DATA ###

# Simulated data ----
# 2 subpopulations
n_obs = 200
n_coeff = 5
rho = 0.2
true_proportions = c(rho, 1-rho)*n_obs
```

```r
intercept_beta = 0.5
beta1 = c(0,5,0,0,1)
beta2 = c(-1,0,4,3,0)

# build covariance matrix
covariance_matrix = matrix(0.5, nrow = length(beta1), ncol = (length(beta1)))
for(i in 1:length(beta1)){
  for(j in 1:length(beta2)){
    covariance_matrix[i,j] = covariance_matrix[i,j]^abs(i - j)
  }
}

dataset = data.matrix(data.frame(mvrnorm(n = n_obs,
                                         mu = rep(0,length(beta1)),
                                         Sigma = covariance_matrix)))
dataset = cbind(1, dataset)
colnames(dataset)[1] = 'intercept'

beta1 = c(intercept_beta,beta1)
beta2 = c(intercept_beta,beta2)

y = c()
# beta1
for(i in 1:true_proportions[1]){
  new_y = dataset[i,]%*%beta1
  y = c(y, new_y)
}

# beta2
for(i in (true_proportions[1]+1):n_obs){
  new_y = dataset[i,]%*%beta2
  y = c(y, new_y)
}

# add error
for(i in 1:length(y)){
  y[i] = runif(1) + y[i]
}



dataset = cbind(y, dataset)

#dim(dataset)

#summary(lm(y ~-1+. ,data=as.data.frame(dataset[1:60,])))
#colnames(dataset)
```

```r
# Initialization variables
TIMES = 5000

n_subpop = 2
x_matrix = as.matrix(dataset[,-1])
```

```r
x_matrix = as.matrix(x_matrix[,order(colnames(x_matrix))])
head(x_matrix)
```

```
##      intercept           X1          X2          X3         X4         X5
## [1,]         1 -0.310975257  0.87035991 -0.77664440 -2.1222032 -0.5849230
## [2,]         1  0.758187063  0.04862675 -0.76196474  1.0721660  1.4740143
## [3,]         1  0.009351107 -0.45659624 -1.11500426  1.3392026  0.6715448
## [4,]         1 -0.632825270  0.01838449 -0.04271129  0.1060108 -1.7353639
## [5,]         1 -1.201932390 -0.11984898 -0.62792268 -0.7607162 -0.3321284
## [6,]         1  1.587747881 -0.05290593 -1.23120303 -0.3008132 -0.5340881
```

```r
initialized = Initialization(x_matrix, y, n_subpop)

theta_array= initialized$theta
#beta_test1 = as.matrix(beta1)
#row.names(beta_test1) = row.names(theta_array$beta[[1]])

#beta_test2 = as.matrix(beta2)
#row.names(beta_test2) = row.names(theta_array$beta[[2]])

#theta_array$beta[[1]] =  beta_test1
#theta_array$beta[[2]] = beta_test2

z_array = initialized$vector_z
history_theta = list()
history_z  = list()

# Run ------------------------------------------------
# 1-4 / 1-2-5
for(i in 1:TIMES){
  #print('-------------------------')
  #print('Iteration:')
  #print(i)
  #print('z')
  new_z = UpdateArrayZ(theta_array, dataset)
  history_z[[i]] = new_z
  z_array[which(new_z == 1),1] = 1
  z_array[which(new_z != 1),1] = 0
  z_array[which(z_array[,1] == 1),2] = 0
  z_array[which(z_array[,1] != 1),2] = 1
  #print('rho')
  theta_array$rho = UpdateRho(z_array)
  #print('subpop')
  covariates = UpdateSubPop(theta_array,z_array, x_matrix, n_subpop)
  #print('W_m inv')
  theta_array = UpdateW_inv(theta_array, covariates, n_subpop)
  #print('sigma')
  #print(theta_array$sigma)
  theta_array = UpdateSigma(theta_array, z_array,covariates)
  #print('beta')
  theta_array = UpdateBeta(z_array,covariates,theta_array)
  #print(theta_array$sigma)
  #print('r')
  theta_array = UpdateR(theta_array, z_array)
```

```
  #print(theta_array$r)
  history_theta[[i]] = theta_array
}




print("number_of_iterations:")
```

```
## [1] "number_of_iterations:"
```

```
print(TIMES)
```

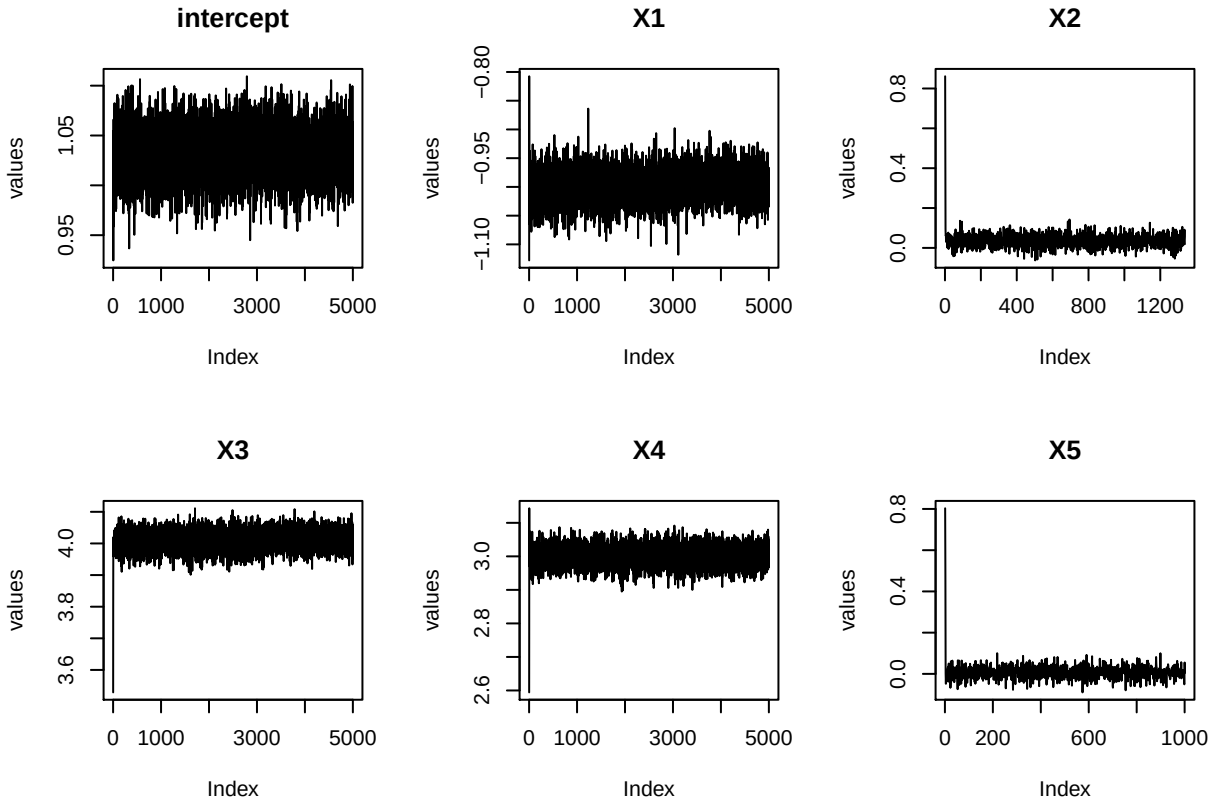
```
## [1] 5000
```

6.2 Diagnostics

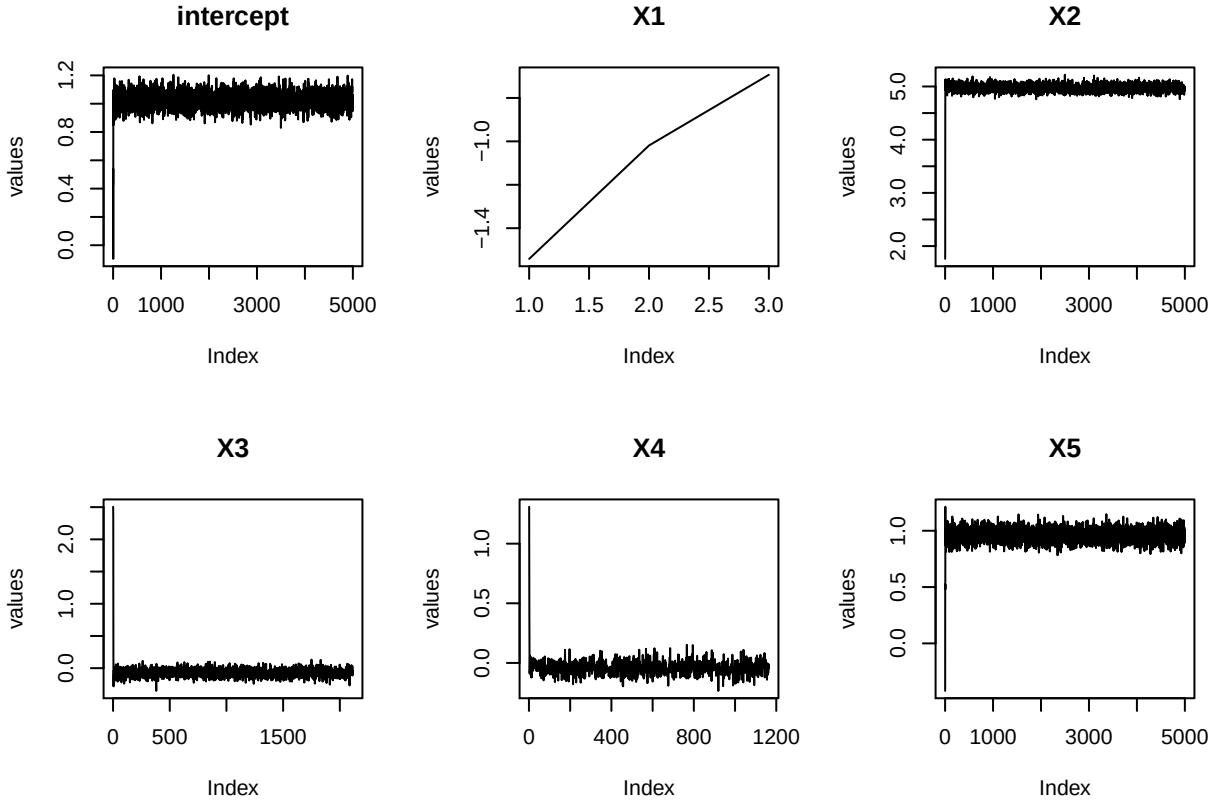
Now, we want to evaluate the performance of our simulation applying the techniques described in the section **5.2**, but first, we look at the distributions of the estimated parameters $\theta_m = (\beta_m, \sigma_m^2, \rho_m, r_m)$

```
### beta ###

n_covariates = 5
summary_parameters = data.frame(matrix(ncol = n_covariates+1, nrow = n_subpop))
colnames(summary_parameters) = row.names(theta_array$old_beta[[1]])

beta_estimates = list()
for(idx_m in 1:n_subpop){
  for(theta in history_theta){
    beta_estimates[[toString(idx_m)]] = rbind(beta_estimates[[toString(idx_m)]], theta$beta[[idx_m]])
  }
}


# plot - lines
for(idx_m in 1:length(beta_estimates)){
  print('Subpopulation n:')
  print(idx_m)
  par(mfrow=c(2,3))
  list_covariates = sort(unique(row.names(beta_estimates[[idx_m]])))
  for(one_cov in list_covariates){
    values = beta_estimates[[idx_m]][which(row.names(beta_estimates[[idx_m]])==one_cov),]
    one_median = median(values)
    summary_parameters[idx_m,one_cov] = one_median
    plot(values,type = 'l', main = one_cov)
  }
}
```

```
## [1] "Subpopulation n:"
## [1] 1
```

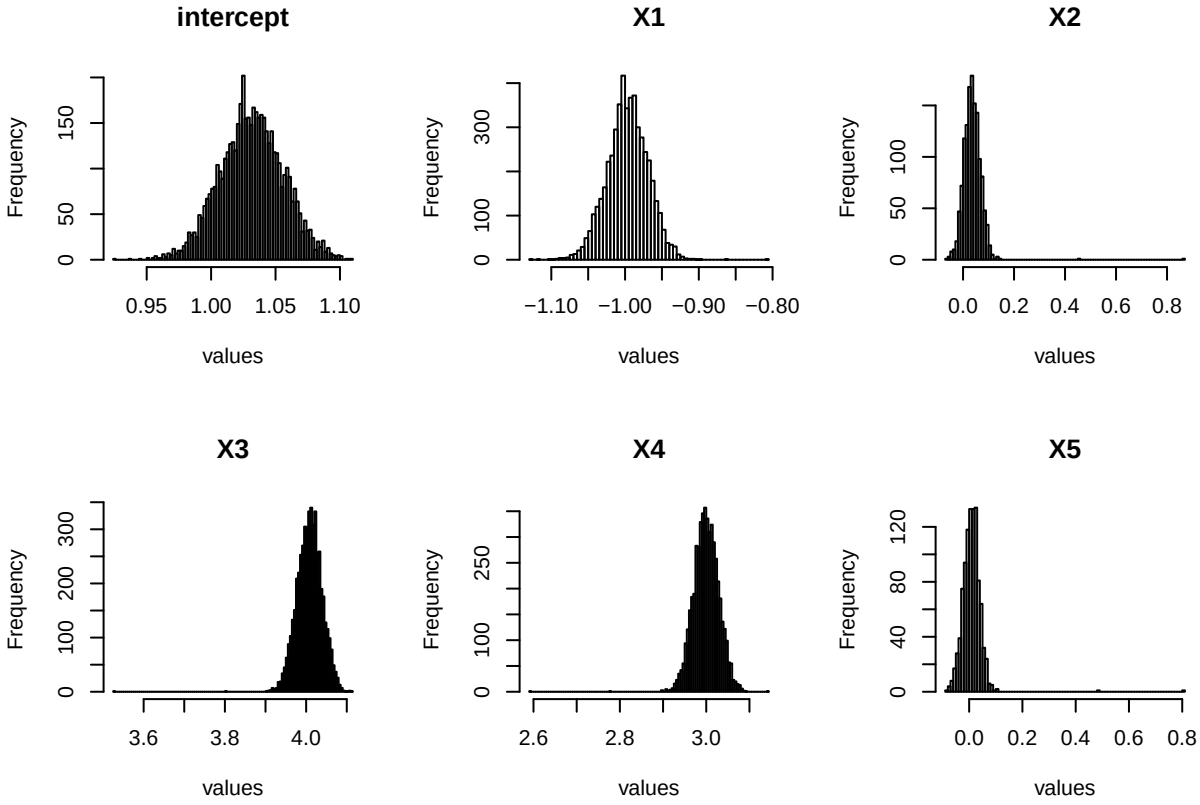



```
## [1] "Subpopulation n:"
## [1] 2
```

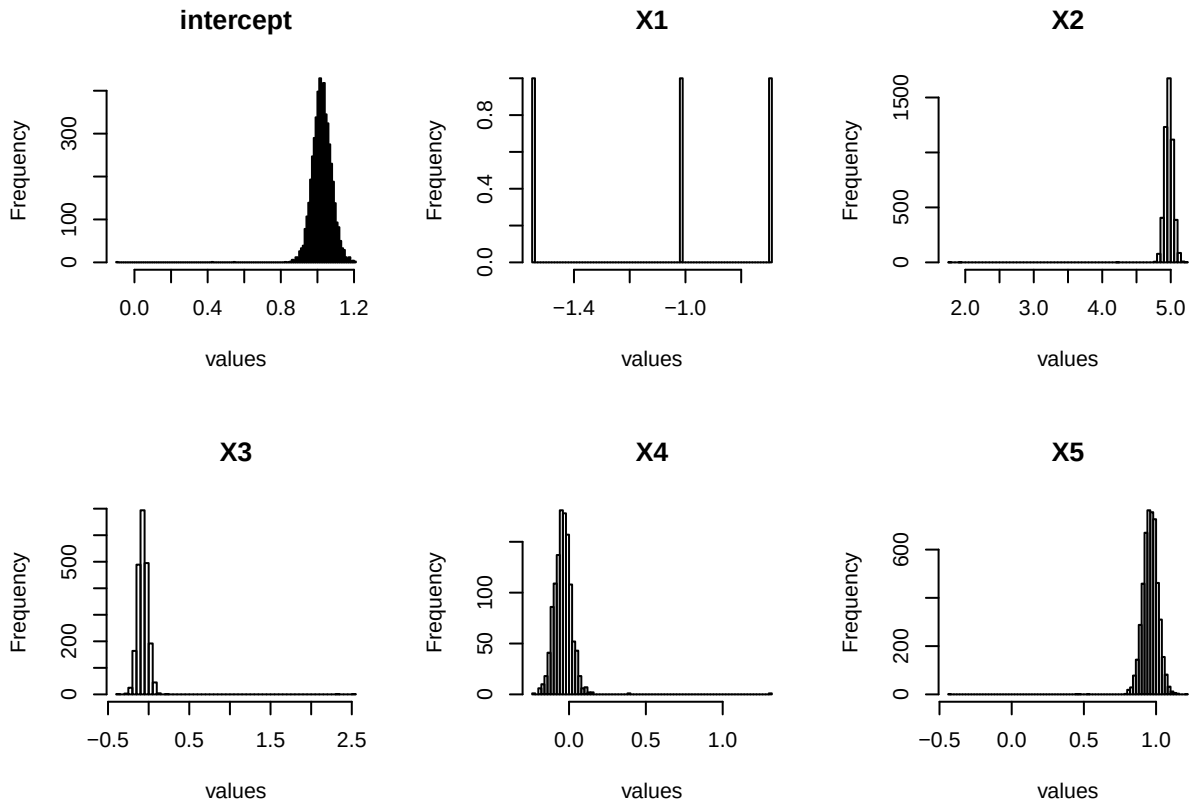


```r
# plot - histogram
for(idx_m in 1:length(beta_estimates)){
  print('Subpopulation n:')
  print(idx_m)
  par(mfrow=c(2,3))
  list_covariates = sort(unique(row.names(beta_estimates[[idx_m]])))
  for(one_cov in list_covariates){
    values = beta_estimates[[idx_m]][which(row.names(beta_estimates[[idx_m]])==one_cov),]
    one_median = median(values)
    hist(values, breaks = 100,main = one_cov)
  }
}
```

```
## [1] "Subpopulation n:"
## [1] 1
```



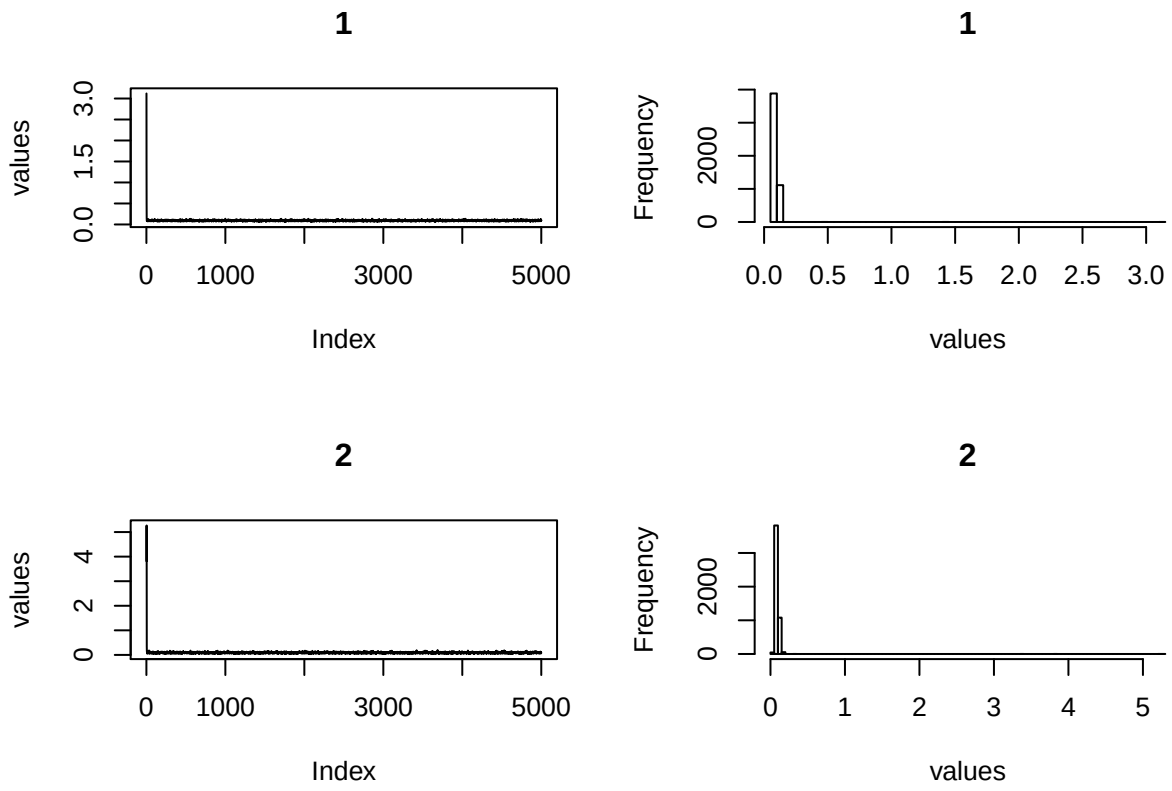

```
## [1] "Subpopulation n:"
## [1] 2
```




```
summary_parameters
```

```
##   intercept         X1         X2          X3          X4          X5
## 1  1.031400 -0.9962881 0.03561616  4.00909854  2.99939094 0.008201954
## 2  1.023761 -1.0189506 4.97292480 -0.07390538 -0.04050115 0.961105944
```

```
sigma_estimates = list()
for(idx_m in 1:n_subpop){
  for(theta in history_theta){
    sigma_estimates[[toString(idx_m)]] = c(sigma_estimates[[toString(idx_m)]], theta$sigma[[idx_m]])
  }
}

# plot - lines and hist
par(mfrow=c(2,2))
for(idx_m in 1:length(sigma_estimates)){
    values = sigma_estimates[[idx_m]]
    plot(values,type = 'l', main = idx_m)
    hist(values, breaks = 100, main = idx_m)
}
```

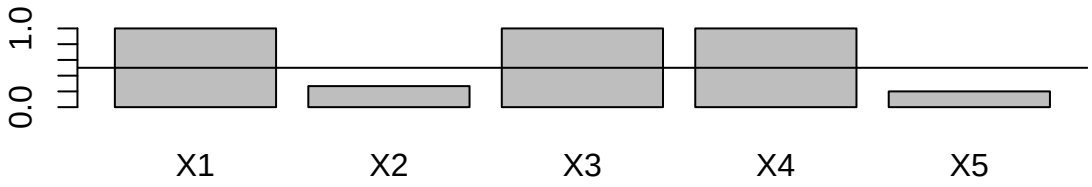


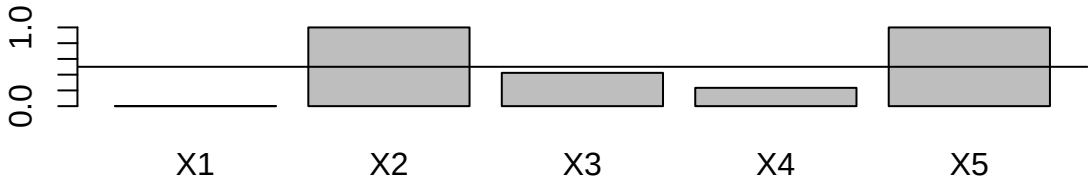
Now, we compute the probabilities related to z and r in order to see which groups and covariates are selected by the algorithm. For the z we just look at the probabilities of the 1st group in order to see if the displayed values look reasonable.

```
### z ###
```

Now, we compute the probabilities of r for each sub-population.

```
prob_r = c()
  for(one_sample_r in history_theta){
   prob_r = cbind(prob_r, one_sample_r$r )
  }

par(mfrow=c(2,1))
for(idx_m in 1:dim(prob_r)[1]){
  one_vec = c()
  list_covariates = sort(unique(colnames(prob_r)))
  list_covariates = list_covariates[2:length(list_covariates)]
  for(one_cov in list_covariates){
    one_p = sum(prob_r[idx_m, which(colnames(prob_r)==one_cov)])/length(history_theta)
    one_vec = c(one_vec, one_p)
  }
  barplot(one_vec, ylim = c(0,1), names.arg = list_covariates)
  abline(0.5, 0)
}
```


1.0
0.0
X1
X2
X3
X4
X5


1.0
0.0
X1
X2
X3
X4
X5