Comparing Mixture Models with EM Algorithm and Gibbs Sampler

STP 540
Spring 2021

Zeyu Yang
Sixue Zhao

April 30, 2021

Introduction

EM algorithm and Gibbs sampler have both been highly successful in analyzing mixture models, especially for parameter estimation. The EM algorithm is an iterative scheme based on maximum likelihood, while Hibbs sampler is an approach of generating random sample from a multivariate distribution (Liang 2009). In practice, it is often difficult to determine which algorithm will work best (Zaheer, Wick et al.). In this project, we focus on the three-component normal mixture model and compare the performance of simulated mixture model using both EM algorithm and Gibbs sampler.

Methods

In our project, we assumed that we know the number of components, and coded up two algorithms based on this assumption. In EM algorithm, we imagine that there is a small number of normal we are mixing together with unequal weights and various components. The first step was constructing initial values for probability at the value 1/3 with three components and mean and standard deviation were simply to choose from uniform distribution at random. In E-step, we got the probability of the ith sample from the jth component and recorded the parameters of last iteration. In M-step, maximizing the log-likelihood by letting its derivation as zero, and here we gave the optimization results directly. We can also set a threshold when the change between last iteration and the next iteration is pretty small, where we think it converges.

Gibbs sampler is Markov Chain that uses full conditional distribution for the proposal distribution for each component. Here we wanted to inference for the mean, standard deviation and probability. Here is the algorithm in outline: 1) sample latent variable I (in our code is z) from an independent multinoulli, 2) sample probability from a Dirichlet, 3) sample mean from an independent normal distribution, and 4) sample standard deviation from an independent inverted chi-squared distribution.

We simulated data with three components, and the mean of each component is 0, 1, 5, the corresponding standard deviations are 1, 0.5, 2 and the probabilities equal to 0.4, 0.4, 0.2, separately. Then we drew 100 simulated samples for each algorithm, and the general simulated data is shown in Figure 1. To illustrate EM algorithm, figure 2 demonstrates the three components (normal distribution) underlying the simulated data.

Figure 1: Histogram of simulated data

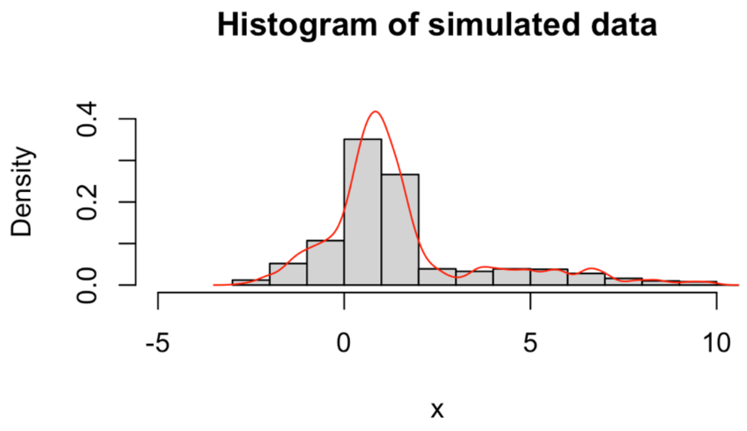
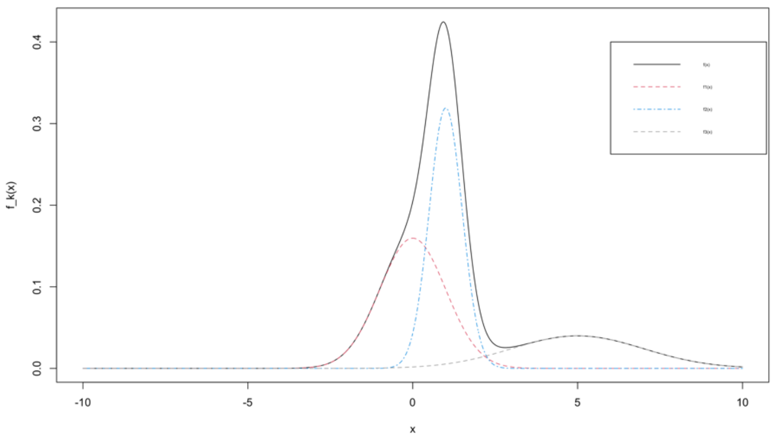


Figure 2: Three normal distribution underlying mixture model



Results

Table 1 and Table 2 showed the results of two algorithms. We can see that those two results were very close to true values, but EM algorithm had a greater performance as its estimations were much closer, especially in probability.

Table 1: Results on EM algorithm

	Component 1	Component 2	Component 3
Mean	-0.026	0.948	5.043

Sigma	0.890	0.528	1.976
Probability	0.365	0.437	0.199

Table 2: Results on Gibbs sampler

	Component 1	Component 2	Component 3
Mean	-0.263	0.954	4.565
Sigma	0.889	0.531	2.213
Probability	0.309	0.473	0.218

We also calculated the mean-square error (MSE) for the 100 observations drawn from the simulated data in order to measure the performance of two algorithms. The smaller MSE we got indicated the better the estimation of algorithm. The results are as follow.

Table 3: MSE on EM algorithm

	Component 1	Component 2	Component 3
MSE(Mean)	0.07	0.02	0.16
MSE(Sigma)	0.09	0.02	0.06
MSE(Probability)	0.02	0.02	0.00

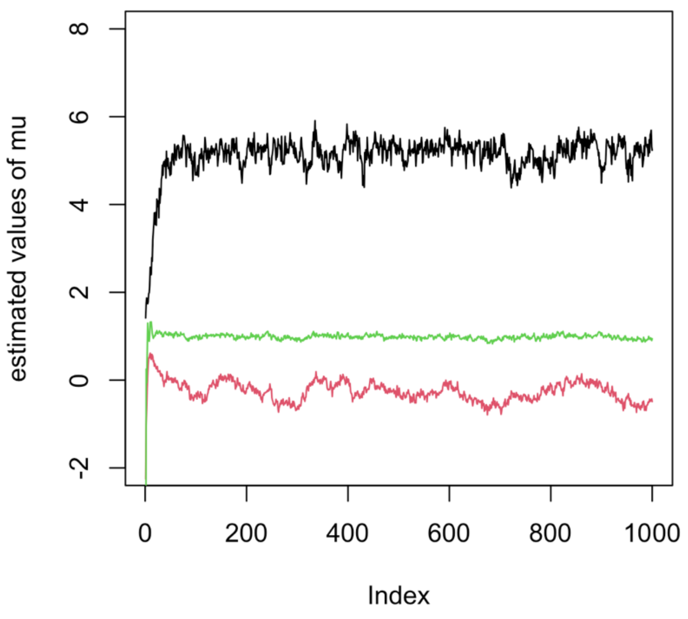
Table 4: MSE on Gibbs sampler

	Component 1	Component 2	Component 3
MSE(Mean)	0.15	0.01	0.42
MSE(Sigma)	0.03	0.01	0.16
MSE(Probability)	0.02	0.01	0.00

In those two tables, we can see that EM algorithm had a better performance in mean. For sigma, sometimes Gibbs sampler won, while other times EM algorithm exceled. We conclude that each algorithm has its own merits.

Try the Gibbs sampler on the data simulated we mentioned above. We can see the first few samples as "burn-in", and then the mean parameters were near the true values. If we drop those "burn-in", the results would be more accurate.

Figure 3: The converge plot of mu in Gibbs Sampler



Discussions

Conclusions

We compared EM algorithm and Gibbs sampler on 100 mixture models drawn from simulated data and found that their performance to be comparable based on the same number of components. They are both good methods for parameter estimation. The EM algorithm doesn't need any prior information, while the Gibbs sampler is more complicated in computing. Summarizing the simulation results and MSE tables, we think they usually get similar results about true values. For estimated means, EM algorithm would be a better choice, while Gibbs sampler is an optimal approach if want to conduct more statistical inference.

In addition, though we didn’t try any vectorization techniques in these two algorithms, we can still explore their computational properties. The Gibbs sampler is sequential and difficult to parallelize, and in our project, it took long time to get the results as well. In contrast, EM algorithm is easy to parallelize, and we got the results in a comparative short time.

Limitations

The results and conclusions derived from this project are subject to some limitations. The following paragraphs will further explain these limitations which include: 1) superficial understanding of the mechanism of the two algorithms, and 2) outliers and unstable result.

Our codes were much more an editing of the lectures from our understandings, but not much at understanding the context. In the project, we assumed that we knew the number of components with simulated data and excluded that part in our codes. The incompleteness made the coded algorithms less functional. We are also unable to get the information of which data points belongs to which component in the EM algorithm.

The other limitation concerns the outliers and unstable result. For example, we drew 100 observations from simulated random sample in the EM algorithm, while our estimates were not completely close to the true values, in fact some of them were outliers that far from the true values. To conclude, we still cannot surely imply that our results can represent the two algorithms as a whole.

Areas of Improvement

In mixture analysis, a key point is that the number of components in the mixture needed for the analysis may depend heavily on the nature of the data (Aitkin 2001). As we mentioned previously, we didn’t code up the latent variable in the EM algorithm in our project. It is yet uncertain that if the lack of latent variable leads to unstable results in our codes, but that is one area of improvement for coding. Additionally, we only consider the case of complete data, which is the easiest way to validate our codes. We could also attempt to include missing values in the dataset and see what happens.

References

Aitkin, M. (2001). "Likelihood and Bayesian analysis of mixtures." Statistical Modelling **1**(4): 287-304.

Liang, L. (2009). On simulation methods for two component normal mixture models under Bayesian approach.

Zaheer, M., et al. "Comparing Gibbs, EM and SEM for MAP Inference in Mixture Models."

Appendix

EM Algorithm

```
EM <- function(samp){
  k <- 3
  n <- 1000
  prob <- matrix(rep(0, k*n), nrow = n)
  weight <- matrix(rep(0, k*n), nrow = n)

  # intial values
  alpha <- c(0.333, 0.333,0.334)
  miu   <- runif(k)
  sigma <- runif(k)

  # EM algorithm
  for (step in 1:200) {
    # E-step
    for (j in 1:k) {
      prob[, j]   <- sapply(samp, dnorm, miu[j], sigma[j])
      weight[, j] <- alpha[j] * prob[, j]
    }
    row_sum <- rowSums(weight)
    prob    <- weight/row_sum
```

```r
    # record the values of parameters
    #oldalpha <- alpha
    #oldmiu   <- miu
    #oldsigma <- sigma

    # M-step
    for (j in 1:k) {
      sum1     <- sum(prob[, j])
      sum2     <- sum(samp*prob[, j])
      alpha[j] <- sum1/n
      miu[j]   <- sum2/sum1
      sum3     <- sum(prob[, j]*(samp-miu[j])^2)
      sigma[j] <- sqrt(sum3/sum1)
    }

    # set threshold
    #threshold <- 1e-5
    #if (sum(abs(alpha - oldalpha)) < threshold &
    #    sum(abs(miu - oldmiu))     < threshold &
    #    sum(abs(sigma - oldsigma)) < threshold) break
    cat('step', step, 'alpha', alpha, 'miu', miu, 'sigma', sigma, '\n')
  }

  alpha1 <- alpha
  sigma1 <- sigma
  for (i in 1:3){
    alpha1[rank(miu)[i]] <- alpha[i]
    sigma1[rank(miu)[i]] <- sigma[i]
  }

  alpha <- alpha1
  sigma <- sigma1
  miu <- sort(miu)
  return(list(miu,sigma,alpha))
```

```
}
```

Simulation EM algorithm

```
rmix = function(n, pi, mu, s){
  z = sample(1:length(pi), prob=pi, size=n, replace=TRUE)
  x = rnorm(n, mu[z], s[z])
  return(x)
}
sim_em <- function(index){
  x <- rmix(n=1000, pi=c(0.4, 0.4, 0.2), mu=c(0, 1, 5), s=c(1, 0.5, 2))
  res <- EM(x)
  return(res)
}

set.seed(1000)
em_res <- llply(1:100, sim_em)

for(i in 1:100){
  if (is.nan(em_res[[101-i]][[2]][[3]])){
    em_res <- em_res[i-101]
  }
}

mu <- function(index, res){
  res[[index]][[1]]
}

sigma <- function(index, res){
  res[[index]][[2]]
}

pi <- function(index, res){
  res[[index]][[3]]
```

```r
}


summary_res <- function(param, data){
  data.frame(Component1 = mean((data[, 1] -param[1])^2),
             Component2 = mean((data[, 2] -param[2])^2),
             Component3 = mean((data[, 3] -param[3])^2))


}


mu_em <- ldply(1:length(em_res), mu, res = em_res)
sigma_em <- ldply(1:length(em_res), sigma, res = em_res)
pi_em <- ldply(1:length(em_res), pi, res = em_res)

em_dat <- rbind(summary_res(c(0, 1, 5), data = mu_em),
                summary_res(c(1, 0.5, 2), data = sigma_em),
                summary_res(c(0.4, 0.4, 0.2), data = pi_em))

rownames(em_dat) <- c("Mean", "Sigma", "Probability")
```

Gibbs Sampler

```r
normalize = function(x){return(x/sum(x))}

sample_z = function(x,pi,mu,sigma){
  #dmat=matrix(0,nrow=3,ncol=1000)
  #for (i in 1:3){
  #  dmat[i,]=(mu[i]-x)/sigma[i]
  #}
  dmat = outer(mu,x,"-") # k by n matrix, d_kj =(mu_k - x_j)
  p.z.given.x = as.vector(pi) * dnorm(dmat,0,sigma)
  p.z.given.x = apply(p.z.given.x,2,normalize) # normalize columns
  z = rep(0, length(x))
  for(i in 1:length(z)){
    z[i] = sample(1:length(pi), size=1,prob=p.z.given.x[,i],replace=TRUE)
  }
  return(z)
}

sample_pi = function(z,k=3){
  counts = colSums(outer(z,1:k,FUN="=="))
  pi = gtools::rdirichlet(1,counts+1)
  return(pi)
}

sample_mu = function(x, z, sigma, mbar=0,tau=1){
  mu = rep(0,3)
  for (i in 1:3){
    sample.size = sum(z==i)
    ybar = ifelse(sample.size==0,0,mean(x[z==i]))
    n<-length((x[z==i]))
    a<-n/sigma[i]^2
    b<-1/tau^2
    mpost = (a*ybar+b*mbar)/(a+b)
    spost = sqrt(1/(a+b))
    mu[i] = rnorm(1,mpost,spost)
  }
  return(mu)
}

sample_sigma = function(x, z, mu, nu=2,lambda=1){
  sigma = rep(0,3)
  for(i in 1:3){
    n<-length((x[z==i]))
    c=1
    x1<-vector()
    for (j in 1:1000){
      if (z[j]==i)
      {x1[c]<-x[j]
      c=c+1}
    }
    S<-sum((x1-mu[i])^2)
    sigma[i] = sqrt((nu*lambda+S)/rchisq(1,nu+n))
  }
  return(sigma)
}


gibbs = function(x,k=3,niter =1000){
  pi = rep(1/k,k) # initialize
  mu = rnorm(k,0,10)
  sigma = rep(1, k)
  z = c(rep(1,333),rep(2,333),rep(3,334))#sample_z(x,pi,mu,sigma)
  res = list(mu=matrix(nrow=niter, ncol=k), pi = matrix(nrow=niter,ncol=k), z
 = matrix(nrow=niter, ncol=length(x)), sigma = matrix(nrow = niter, ncol =
k))
  res$mu[1,]=mu
  res$pi[1,]=pi
  res$z[1,]=z
  res$sigma[1, ] = sigma
  for(i in 2:niter){
    sigma = sample_sigma(x, z,  mu, nu=2, lambda=1)
    mu = sample_mu(x,z,sigma,mbar=0,tau=1)
    z = sample_z(x,pi,mu,sigma)
    pi = sample_pi(z,k)
    res$mu[i,] = mu
    res$pi[i,] = pi
    res$z[i,] = z
    res$sigma[i, ] = sigma
  }
  return(res)
}
```

Simulation Gibbs Sampler algorithm

```
rmix = function(n, pi, mu, s){
  z = sample(1:length(pi), prob=pi, size=n, replace=TRUE)
```

```r
  x = rnorm(n, mu[z], s[z])
  return(x)
}
sim_gibbs <- function(n){
  x <- rmix(n=1000, pi=c(0.4, 0.4, 0.2), mu=c(0, 1, 5), s=c(1, 0.5, 2))
  res <- gibbs(x,3)
  post_mu <- c(res$mu[1000, 1], res$mu[1000, 2], res$mu[1000, 3])
  post_sigma <- c(res$sigma[1000, 1], res$sigma[1000, 2], res$sigma[1000, 3])
  post_pi <- c(res$pi[1000, 1], res$pi[1000, 2], res$pi[1000, 3])
  post_pi1 <- post_pi
  post_sigma1 <- post_sigma
  for(i in 1:3){
    post_pi1[rank(post_mu)[i]] <- post_pi[i]
    post_sigma1[rank(post_mu)[i]] <- post_sigma[i]
  }

  post_pi <- post_pi1
  post_sigma <- post_sigma1
  post_mu <- sort(post_mu)
  return(list(post_mu, post_sigma, post_pi))
}

set.seed(1000)
gibbs_res <- llply(1:100, sim_gibbs)

mu <- function(index, res){
  res[[index]][[1]]
}

sigma <- function(index, res){
  res[[index]][[2]]
}

pi <- function(index, res){
  res[[index]][[3]]
}

summary_res <- function(param, data){
  data.frame(Component1 = mean((data[, 1] -param[1])^2),
             Component2 = mean((data[, 2] -param[2])^2),
             Component3 = mean((data[, 3] -param[3])^2))

}

mu_gibbs <- ldply(1: 100, mu, res = gibbs_res)
sigma_gibbs <- ldply(1: 100, sigma, res = gibbs_res)
pi_gibbs <- ldply(1: 100, pi, res = gibbs_res)

gibbs_dat <- rbind(summary_res(c(0, 1, 5), data = mu_gibbs),
                summary_res(c(1, 0.5, 2), data = sigma_gibbs),
                summary_res(c(0.4, 0.4, 0.2), data = pi_gibbs))

rownames(gibbs_dat) <- c("Mean", "Sigma", "Probability")
```