HMM-VB

Jianghao Li

3/29/2017

Supppose a vector is partitioned in to T variable blocks, ie $x = (x^{(1)}, x^{(2)}, ..., x^{(T)}$. For each t, let S_t denote the set of Gaussian mixture components at time t. The forward probability for the k^{th} state in time t is defined as:

$$\alpha_k(x,t) = P(x^{(1)}, x^{(2)}, ..., x^{(t)}, s_t = k), k \in S_t$$

It can be calculated using the recursive formula:

$$\alpha_k(x,1) = \pi_k \phi(x^{(1)}|\mu_k^{(1)}, \Sigma_k^{(1))}, k \in S_t$$
$$\alpha_k(x,t) = \phi(x^{(t)}|\Sigma_k^{(t)}, \Sigma_k^{(t)}) \sum_{l \in S_{t-1}} \alpha_l(x, t-1) a_{l,k}^{(t-1)}, 1 < t \leq T, k \in S_t$$

where $\pi_k = P(s_1 = k)$. The code to calculate forward probability is shown below. What is noticeable is that argument x is a vector, corresponding to one single observation.

```
forw = function(x, mu, sigma, pi, a){
  alpha = vector("list", Ti)
  alpha[[1]] = rep(0, S[1])
    for(k in 1:S[1])
    alpha[[1]][k]=pi[k]*dmvnorm(x[1:B[1]],mu[[1]][[k]], sigma[[1]][[k]])

  for(t in 2:Ti){
    alpha[[t]] = rep(0, S[t])
    for(k in 1:S[t])
    alpha[[t]][k] = dmvnorm(x[(B[t-1]+1):B[t]],mu[[t]][[k]],
                    sigma[[t]][[k]])*sum(alpha[[t-1]]*a[[t-1]][,k])
  }
  return(alpha)
}
```

Next, backward probability can be defined as

$$\beta_k(x,t) = P(x^{(t+1)}, x^{(t+2)}, ..., x^{(T)}|s_t = k)$$

And it can be solved recursively by

$$\beta_k(x,T) = 1$$
$$\beta_k(x,t) = \sum_{l \in S_{t+1}} a_{l,k}^{(t)} \phi(x^{(t+1)}|\mu_l^{(t+1)}, \Sigma_l^{(t+1)}) \beta_l(x, t-1)$$

The function in R is shown below. x is also a vector here

```
backw = function(x, mu, sigma,a){
  beta = vector("list", Ti)
  beta[[Ti]]=rep(1, S[Ti])
  for(t in Ti-1:1){
```

```
    beta[[t]] = rep(0, S[t])
    for(k in 1:S[t])
      for(l in 1:S[t+1])
      beta[[t]][k] = beta[[t]][k] + (dmvnorm(x[(B[t]+1):B[t+1]],
                     mu[[t+1]][[l]], sigma[[t+1]][[l]])*beta[[t+1]][l]*a[[t]][k,l])
  }
  return(beta)
}
```

Then two posteria probabilities can be calculated by:

$$L_k(x,t) = P(s_t = k|x) = \frac{\alpha_k(x,t)\beta_k(x,t)}{P(x)}$$

$$H_{k,l}(x,t) = P(s_t = k, s_{t+1} = l|x) = \frac{1}{P(x)}\alpha_k(x,t)a_{k,l}^{(t)}\phi(x^{(t+1)}|\mu_l^{(t+1)}, \Sigma_l^{(t+1)})\beta_l(x,t)$$

where $P(x) = \sum_{k \in S_t} \alpha_k(x,t)\beta_t(x,t)$

```
lik = function(x, mu, sigma, pi,a){
    lik = vector("list", Ti)
    alpha = forw(x, mu, sigma, pi,a)
    beta = backw(x, mu, sigma,a)
    for(t in 1:Ti){
      lik[[t]] = rep(0, S[t])
      for (k in 1:S[t]){
        lik[[t]][k] = alpha[[t]][k]*beta[[t]][k]/sum(alpha[[t]]*beta[[t]])
      }
    }
  return (lik)
}

h = function(x, mu, sigma, pi,a){
    H = vector("list", Ti-1)
    alpha = forw(x, mu, sigma, pi,a)
    beta = backw(x, mu, sigma,a)
    for(t in 1:(Ti-1)){
      H[[t]] = matrix(0, S[t], S[t+1])
      for (k in 1: S[t]){
        for(l in 1:S[t+1]){
          H[[t]][k,l] = alpha[[t]][k]*a[[t]][k,l]*dmvnorm(x[(B[t]+1):B[t+1]],mu[[t+1]][[l]],
                          sigma[[t+1]][[l]])*beta[[t+1]][l]/sum(alpha[[t]]*beta[[t]])
        }
      }
    }
  return(H)
}
```

Next is the Baum-Welch algorithm to estimate all the parameters used above. This is a EM algorithm where the E_step is computing $L_k(x,t)$ and $H_{k,l}(x,t)$. and the M-step is updating the parameters by:

$$
\begin{aligned}
\mu_k^{(t)} &= \frac{\sum_{i=1}^n L_k(x_i,t) x_i^{(t)}}{\sum_{i=1}^n L_k(x_i,t)} \\
\Sigma_k^{(t)} &= \frac{\sum_{i=1}^n L_k(x_i,t)\left(x_i^{(t)}-\mu_k^{(t)}\right)\left(x_i^{(t)}-\mu_k^{(t)}\right)'}{\sum_{i=1}^n L_k(x_i,t)} \\
a_{k,l}^{(t)} &= \frac{\sum_{i=1}^n H_{k,l}(x_i,t)}{\sum_{i=1}^n L_k(x_i,t)} \\
\pi_k &\propto \sum_{i=1}^n L_k(x_i,1)
\end{aligned}
$$

The code is given below. Notice that this function works on all the observations, therefore argument data is a dataset containing all the observations.

```
BW = function(data, mu0, sigma0, pi, a,tol = 1e-4, maxit = 10){
  mu1 = mu0
  sigma1 = sigma0
  a1 = a
  pi1 = pi
  n=dim(data)[1]
#calculate likelihood for each observation in the data set
  for(j in 1:maxit){
    l_temp = vector("list", Ti)
    h_temp = vector("list", Ti-1)
    for(t in 1:Ti){
      l_temp[[t]] = vector("list", S[t])
      if(t < Ti)
        h_temp[[t]] = matrix(0, S[t], S[t+1])
        for( k in 1:S[t]){
          l_temp[[t]][[k]] = rep(0, n)
          for( i in 1:n){
            l1 = lik(data[i,], mu1, sigma1, pi1,a1)
            h1 = h(data[i,], mu1, sigma1, pi1,a1)
            l_temp[[t]][[k]][i] = l1[[t]][k]
            if(t < Ti && k ==1 )
              h_temp[[t]] = h1[[t]] + h_temp[[t]]
          }
        }
    }
#Use the likelihood to iteratively estimate the parameters
    for(t in 1:Ti){
      for(k in 1:S[t]){
        if(t==1) data_temp = data[,1:B[1]]
        else data_temp = data[,(B[t-1]+1):B[t]]
        mu1[[t]][[k]] = apply(l_temp[[t]][[k]]*data_temp, 2, sum)/sum(l_temp[[t]][[k]])
        sigma1[[t]][[k]] = (t(data_temp - mu1[[t]][[k]]))%*%(l_temp[[t]][[k]]*
                             (data_temp - mu1[[t]][[k]]))/sum(l_temp[[t]][[k]])
        if(t ==1) pi1[k] = sum(l_temp[[1]][[k]])
        if(t < Ti)
          a1[[t]][k,] = h_temp[[t]][k,]/sum(l_temp[[t]][[k]])
      }
    }
```

```
    pi1 = pi1/sum(pi1)
  }
  return(list(mu = mu1, sigma = sigma1, pi = pi1, a = a1))
}
```

Finally comes to the Modal Baum-Welch algorithm. In the M-step of this EM algorithm, the mode is updated by

$$x^{[r+1]} = \left(\sum_{k \in S_t} L_k(x^{[r]}, t) (\Sigma_k^{(t)})^{-1} \right)^{-1} \left(\sum_{k \in S_t} L_k(x^{[r]}, t) (\Sigma_k^{(t)})^{-1} \mu_k^{(t)} \right)$$

And in the E-step calculate $L_k(x^{[r]}, t)$. The purpose of this function is to calculate the probability of each variable block in an observation belongs to each component in the GMM. The function in R is below. The argument x is also a vector.

```
MBW = function(x, mu, sigma, pi, a, tol = 1e-4, maxit = 100){
  mode1 = x
  mode2 = x
  for (i in 1:maxit){
    l1 = lik(mode1, mu, sigma, pi,a)
    for(t in 1:Ti){
      if (t == 1) {
        num = rep(0, B[1])
        den = matrix(0, B[1], B[1])
      }
      else {
        num = rep(0, B[t] - B[t-1])
        den = matrix(0, B[t] - B[t-1],B[t] - B[t-1])
      }
      for(k in 1:S[t]){
        den = den + l1[[t]][k]*solve(sigma[[t]][[k]])
        num = num + l1[[t]][k]*solve(sigma[[t]][[k]])%*%mu[[t]][[k]]
      }
      if(t==1)
        mode2[0:B[1]] = solve(den)%*%num
      else mode1[(B[t-1]+1):B[t]] = solve(den)%*%num
    }
    mode1 = mode2
  }
  return(l1)
}
```