Assignment_3_bonus_8259_binfeng2

Bin Feng

```
library(HMM)
```

```
# Generate Samples from an HMM
T = 200
A0 = rbind(c(0.8, 0.2),
           c(0.2, 0.8))
B0 = rbind(c(0.1, 0.2, 0.7),
           c(0.4, 0.3, 0.3))
w0 = c(0.5, 0.5)
para0 = list(mz = 2, mx = 3, w = w0,
             A = A0, B = B0)
genHMM = function(para, n){
  # return n samples from HMM with parameter = para
  z = para$mz
  mx = para$mx
  w = para$w
  A = para$A
  B = para$B
  Z = rep(0, n)
  X = rep(0, n)

  ## YOUR CODE: generate Z[1]
  Z[1] = sample(c("A","B"), 1, replace = TRUE, prob = w0)
  for(i in 2:n)
    ## YOUR CODE: generate Z[i]
    if(Z[i-1] == "A"){
      Z[i] = sample(c("A","B"), 1, replace = TRUE, prob = A[1,])
    } else if (Z[i-1] == "B"){
      Z[i] = sample(c("A","B"), 1, replace = TRUE, prob = A[2,])
    } else{
      print("Error: neither A or B")
    }

  for(i in 1:n)
    ## YOUR CODE: generate X[i]
    if(Z[i] == "A"){
      X[i] = sample(c(1, 2, 3), 1, replace = TRUE, prob = B[1,])
    } else if(Z[i] == "B"){
      X[i] = sample(c(1, 2, 3), 1, replace = TRUE, prob = B[2,])
    }
  return(X)
}

data = genHMM(para0, T)
```

```
# The Baum-Welch (i.e., EM) Algorihtm
forward.prob = function(x, para){
  # Output the forward probability matrix alp
  # alp: T by mz, (t, i) entry = P(x_{1:t}, Z_t = i)
```

```
  T = length(x)
  mz = para$mz
  A = para$A
  B = para$B
  w = para$w
  alp = matrix(0, T, mz)

  # fill in the first row of alp
  alp[1, ] = w * B[, x[1]]
  # Recursively compute the remaining rows of alp
  for(t in 2:T){
    tmp = alp[t-1, ] %*% A
    alp[t, ] = tmp * B[, x[t]]
    }
  return(alp)
}

backward.prob = function(x, para){
  # Output the backward probability matrix beta
  # beta: T by mz, (t, i) entry = P(x_{1:t}, Z_t = i)
  T = length(x)
  mz = para$mz
  A = para$A
  B = para$B
  w = para$w
  beta = matrix(1, T, mz)

  # The last row of beta is all 1.
  # Recursively compute the previous rows of beta
  for(t in (T-1):1){
    tmp = as.matrix(beta[t+1, ] * B[, x[t+1]])  # make tmp a column vector
    beta[t, ] = t(A %*% tmp)
    }
  return(beta)
}


BW.onestep = function(x, para){
  # Input:
  # x: T-by-1 observation sequence
  # para: mx, mz, and current para values for
  #    A: initial estimate for mz-by-mz transition matrix
  #    B: initial estimate for mz-by-mx emission matrix
  #    w: initial estimate for mz-by-1 initial distribution over Z_1
  # Output the updated parameters after one iteration
  # We DO NOT update the initial distribution w

  T = length(x)
  mz = para$mz
  mx = para$mx
  A = para$A
  B = para$B
  w = para$w
```

```
  alp = forward.prob(x, para)
  beta = backward.prob(x, para)

  myGamma = array(0, dim=c(mz, mz, T-1))
  ## YOUR CODE:
  ## Compute gamma_t(i,j) P(Z[t] = i, Z[t+1]=j),
  ## for t=1:T-1, i=1:mz, j=1:mz,
  ## which are stored an array, myGamma
  for(i in 1:T-1){
    for(j in 1: ncol(A)){
      for(k in 1:ncol(A)){
        myGamma[j,k,i] = alp[i,j] * A[j,k] * B[k,x[i+1]] * beta[i+1,k]
      }
    }
  }

  # M-step for parameter A
  A = rowSums(myGamma, dims = 2)
  A = A/rowSums(A)
  # M-step for parameter B
  tmp = apply(myGamma, c(1, 3), sum)  # mz-by-(T-1)
  tmp = cbind(tmp, colSums(myGamma[, , T-1]))
  for(l in 1:mx){
    B[, l] = rowSums(tmp[, which(x==l)])
    }
  B = B/rowSums(B)

  para$A = A
  para$B = B
  return(para)
}

myBW = function(x, para, n.iter = 100){
  # Input:
  # x: T-by-1 observation sequence
  # para: initial parameter value
  # Output updated para value (A and B; we do not update w)

  for(i in 1:n.iter){
    para = BW.onestep(x, para)
  }
  return(para)
}
```

```
# The Viterbi Algorithm
myViterbi = function(x, para){
  # Output: most likely sequence of Z (T-by-1)
  T = length(x)
  mz = para$mz
  A = para$A
  B = para$B
  w = para$w
  log.A = log(A)
  log.w = log(w)
```

```
  log.B = log(B)

  # Compute delta (in log-scale)
  delta = matrix(0, T, mz)
  # fill in the first row of delta
  delta[1, ] = log.w + log.B[, x[1]]

  ## YOUR CODE:
  ## Recursively compute the remaining rows of delta
  for(i in 2:T){
    delta[i,1] = max(delta[i-1,] + log.A[,1]) + log.B[1,x[i]]
    delta[i,2] = max(delta[i-1,] + log.A[,2]) + log.B[2,x[i]]
  }

  # Compute most prob sequence Z
  Z = rep(0, T)
  # start with the last entry of Z
  Z[T] = which.max(delta[T, ])
  Z[T] = as.numeric(Z[T])
  ## YOUR CODE:
  ## Recursively compute the remaining entries of Z
  for(i in (T-1):1){
    Z[i] = which.max(delta[i, ] + log.A[,Z[i+1]])
  }
  return(Z)
}
```

```
# Test accuracy
data = genHMM(para0, T)
mz = 2
mx = 3
ini.w = rep(1, mz); ini.w = ini.w / sum(ini.w)
ini.A = matrix(1, 2, 2); ini.A = ini.A / rowSums(ini.A)
ini.B = matrix(1:6, 2, 3); ini.B = ini.B / rowSums(ini.B)
ini.para = list(mz = 2, mx = 3, w = ini.w,
                A = ini.A, B = ini.B)

myout = myBW(data, ini.para, n.iter = 100)
myout.Z = myViterbi(data, myout)
myout.Z[myout.Z==1] = 'A'
myout.Z[myout.Z==2] = 'B'

# use package
hmm0 =initHMM(c("A", "B"), c(1, 2, 3),
              startProbs = ini.w,
              transProbs = ini.A,
              emissionProbs = ini.B)
Rout = baumWelch(hmm0, data, maxIterations=100, delta=1E-9, pseudoCount=0)
Rout.Z = viterbi(Rout$hmm, data)

# Compare estimation for transition matrix A
myout$A
```

```
##           [,1]      [,2]
```

```
## [1,] 0.4797081 0.5202919
## [2,] 0.4623306 0.5376694
```

```
Rout$hmm$transProbs
```

```
##     to
## from         A         B
##    A 0.4797081 0.5202919
##    B 0.4623306 0.5376694
```

```
# Compare estimation for emission matrix B
myout$B
```

```
##           [,1]      [,2]      [,3]
## [1,] 0.1810356 0.2537732 0.5651912
## [2,] 0.2735772 0.2560918 0.4703310
```

```
Rout$hmm$emissionProbs
```

```
##       symbols
## states         1         2         3
##      A 0.1810356 0.2537732 0.5651912
##      B 0.2735772 0.2560918 0.4703310
```

```
# Compare the most probable Z-sequence
cbind(Rout.Z, myout.Z)[c(1:10, 180:200), ]
```

```
##       Rout.Z myout.Z
##  [1,] "A"    "A"
##  [2,] "A"    "A"
##  [3,] "B"    "B"
##  [4,] "B"    "B"
##  [5,] "B"    "B"
##  [6,] "A"    "A"
##  [7,] "A"    "A"
##  [8,] "B"    "B"
##  [9,] "B"    "B"
## [10,] "B"    "B"
## [11,] "A"    "A"
## [12,] "B"    "B"
## [13,] "B"    "B"
## [14,] "B"    "B"
## [15,] "A"    "A"
## [16,] "A"    "A"
## [17,] "B"    "B"
## [18,] "B"    "B"
## [19,] "B"    "B"
## [20,] "B"    "B"
## [21,] "B"    "B"
## [22,] "B"    "B"
## [23,] "B"    "B"
## [24,] "B"    "B"
## [25,] "B"    "B"
## [26,] "A"    "A"
## [27,] "A"    "A"
## [28,] "B"    "B"
## [29,] "B"    "B"
```

```
## [30,] "B"    "B"
## [31,] "B"    "B"
```

```
sum(Rout.Z != myout.Z)
```

```
## [1] 0
```

Codes for a hidden Markov model (HMM) with two hidden states and three observed states are shown below. The estimated transition matrix, emission matrix, and the most probable Z-sequence are outputed above as well and compared with the results from the HMM package. Note that all these three results from my code are identical compared with the ones from HMM package.