Project 2

Richard Affolter

4 3 2021

Problem 4: Responsibilities and prior for biased coins

$$
\begin{aligned}
P(C_i \mid D_j) &= \frac{P(C_i, D_j)}{\sum\limits_i P(C_i, D_j)} \\
&= \frac{P(D_j \mid C_i)P(C_i)}{\sum\limits_i P(D_j \mid C_i)P(C_i)}
\end{aligned}
$$

For tossing a coin C_i with probability for heads θ_i $n = 10$ times in total we get

$$
P(D_j \mid C_i) = \binom{n}{k_j}\theta_i^{k_j} \cdot (1-\theta_i)^{n-k_j}
$$

where k_j is the number of heads in heads in trial j.

We can simplify the expression since the binomial coefficient does not depend on i and we know that in this case the prior $P(C_i) = \lambda, i = 1, 2$ for both coins is 0.5.

$$
\begin{aligned}
&\frac{P(D_j \mid C_i)P(C_i)}{\sum\limits_i P(D_j \mid C_i)P(C_i)} \\
&= \frac{\binom{n}{k_j}\theta_i^{k_j} \cdot (1-\theta_i)^{n-k_j}\lambda}{\sum\limits_i \binom{n}{k_j}\theta_i^{k_j} \cdot (1-\theta_i)^{n-k_j}\ \lambda} \\
&= \frac{\theta_i^{k_j} \cdot (1-\theta_i)^{n-k_j}}{\sum\limits_i \theta_i^{k_j} \cdot (1-\theta_i)^{n-k_j}}
\end{aligned}
$$

$$
\begin{aligned}
\gamma_{1,1} = P(C_1 \mid D_1) &= \frac{0.6^4 \cdot 0.4^6}{0.6^4 \cdot 0.4^6 + 0.4^4 \cdot 0.6^6} = 0.308 \\
\gamma_{1,2} = P(C_2 \mid D_1) &= \frac{0.4^4 \cdot 0.6^6}{0.4^4 \cdot 0.6^6 + 0.6^4 \cdot 0.4^6} = 0.692 \\
&= 1 - P(C_1 \mid D_1)
\end{aligned}
$$

$$
\begin{aligned}
\gamma_{2,1} = P(C_1 \mid D_2) &= \frac{0.6^8 \cdot 0.4^2}{0.6^8 \cdot 0.4^2 + 0.4^8 \cdot 0.6^2} = 0.919 \\
\gamma_{2,2} = P(C_2 \mid D_2) &= \frac{0.4^8 \cdot 0.6^2}{0.4^8 \cdot 0.6^2 + 0.6^8 \cdot 0.4^2} = 0.081 \\
&= 1 - P(C_1 \mid D_2)
\end{aligned}
$$

To update the mixture weights:

$$
\begin{aligned}
\hat{\lambda}_k &= \frac{1}{N}\sum_{i=1}^{N}\gamma_{i,k} \\
P(C_1) = \hat{\lambda}_1 &= \frac{1}{2}(0.308 + 0.919) = 0.6135 \\
P(C_2) = \hat{\lambda}_2 &= \frac{1}{2}(0.692 + 0.081) = 0.3865 \\
&= 1 - P(C_1)
\end{aligned}
$$

Data Analysis Problem 5: Learning a mixture model for two biased coins

(a) Load the data stored in the file 'cointoss.csv'

```
data = read.csv(file = "cointoss.csv", header = TRUE)

# count the number of heads for each trial
trials = colSums(data)

# count the number of tosses per trial
L = dim(data)[1]

# count the number of trials
N = dim(data)[2]
```

```
trials = c(5,9,8,4,7)
N = 5
L = 10
```

(b) Start with random priors (mixture weights) (e.g., λ= (0.5, 0.5)), and probabilities for heads (= 1) for each of the coins.

```
#initialize the weights and the probabilites for heads
lambda_init = c(0.5,0.5)
theta_init = c(0.6,0.5)
```

(c) E-step: use the prior and coin probabilities to compute the joint probability of the observed and hidden data. Compute the responsibilities γ and the log of the observed likelihood.

```
# P(C_i,D_j) meaning coin i and trial j
get_joint_prob = function(i,j){
  return (lambda[i] * dbinom(trials[j],L,theta[i]))
}

get_log_cond_prob = function(i,j){
  return (log(dbinom(trials[j],L,theta[i])))
}
```

```r
# fill a matrix (2xN) with all joint probabilites
joint_prob = outer(1:2,1:N, FUN = get_joint_prob)

# fill a matrix (2xN) with all log conditional probabilites
log_cond_prob = outer(1:2,1:N, FUN = get_log_cond_prob)

# compute the marginal of D_j
marginal = colSums(joint_prob)

# compute the responsibility of component i for observation j
get_responsibility = function(i,j){
  return (joint_prob[i,j]/marginal[j])
}

# fill a matrix (2xN) with all responsibilites
responsibilites = outer(1:2, 1:N, FUN = Vectorize(get_responsibility))


# compute the log of the observed likelihood like on slide 18
part1 = sum(rowSums(responsibilites)*log(lambda)) # fist part of the sum
part2 = sum(responsibilites * log_cond_prob) # second part of the sum
l_obs = part1 + part2
```

(d) M-step: use the responsibilities to recompute the priors and the probability of heads (1) of each coin.

```r
# maximize w.r.t. lambda
lambda = rowSums(responsibilites)/N

# maximize w.r.t. theta
temp = (responsibilites %*% trials)/
  (responsibilites %*% trials + responsibilites %*% (L-trials))
theta = drop(temp) #get rid of the unneeded dimensions
```

(e) Reiterate over E- and M-step until convergence of the likelihood.

With the functions defined above:

```r
EM = function(data, theta, lambda){

  # count the number of heads for each trial
  trials = colSums(data)

  # count the number of tosses per trial
  L = dim(data)[1]

  # count the number of trials
  N = dim(data)[2]

  # initialize log likelihood of previous iteration
```

```r
old_l_obs = 0

iteration = 0

progress = 1

while(progress > 1e-7 && iteration < 1e6){

#----------- E-step -----------------------------------

# P(C_i,D_j) meaning coin i and trial j
get_joint_prob = function(i,j){
  return (lambda[i] * dbinom(trials[j],L,theta[i]))
}

get_log_cond_prob = function(i,j){
  return (log(dbinom(trials[j],L,theta[i])))
}

# fill a matrix (2xN) with all joint probabilites
joint_prob = outer(1:2,1:N, FUN = get_joint_prob)

# fill a matrix (2xN) with all log conditional probabilites
log_cond_prob = outer(1:2,1:N, FUN = get_log_cond_prob)

# compute the marginal of D_j
marginal = colSums(joint_prob)

# compute the responsibility of component i for observation j
get_responsibility = function(i,j){
  return (joint_prob[i,j]/marginal[j])
}

# fill a matrix (2xN) with all responsibilites
responsibilites = outer(1:2, 1:N, FUN = Vectorize(get_responsibility))


# compute the log of the observed likelihood like on slide 18
part1 = sum(rowSums(responsibilites)*log(lambda)) # fist part of the sum
part2 = sum(responsibilites * log_cond_prob) # second part of the sum
l_obs = part1 + part2

#----------- M-step ---------------------------------------------

# maximize w.r.t. lambda
lambda = rowSums(responsibilites)/N

# maximize w.r.t. theta
temp = (responsibilites %*% trials)/
  (responsibilites %*% trials + responsibilites %*% (L-trials))
theta = drop(temp) #get rid of the unneeded dimensions

#------ update conditions for loop termination-----------------
```

```
  progress = abs(l_obs - old_l_obs)
  old_l_obs = l_obs
  iteration = iteration + 1
  }
  return(list(lambda = lambda, theta = theta, gamma = responsibilites,
              log_likelihood = l_obs, progress=progress, iteration = iteration))
}
```

We can start the EM several times independently with different priors and choose the result with the highest likelihood

```
set.seed(123)

best_likelihood = -1000

# run the algorithm 100 times
for(i in 1:100){
  # generate random initialization for theta and lambda
  random = runif(2)
  theta_init = c(random[1], 1-random[1])
  lambda_init = c(random[2], 1-random[2])

  current_result = EM(data = data, theta = theta_init, lambda = lambda_init)


  if(current_result$log_likelihood > best_likelihood){
    best_result = current_result
    best_likelihood = current_result$log_likelihood
  }
}
```

(f) Print the probability of heads for each coin and the mixture weights λ. Plot a heatmap of the responsibilities. How many trials belong to each coin?

```
for(i in 1:2){
  cat(
    paste0("Probability of heads for coin", i, ": ",
           round(best_result$theta[i],digits = 6), "\n") )
}
```

```
## Probability of heads for coin1: 0.451041
## Probability of heads for coin2: 0.572846
```

```
cat("\n Mixture weights: (", round(best_result$lambda,digits = 6),")")
```

```
##
##  Mixture weights: ( 0.841068 0.158932 )
```

So we have $\theta_1 = 0.451041$ and $\theta_2 = 0.572846$.

The mixture weights are $\lambda_1 = 0.841068$ and $\lambda_2 = 1 - \lambda_1 = 0.158932$

```r
library(ComplexHeatmap)
rownames(best_result$gamma) = paste("coin",1:2)
Heatmap(best_result$gamma,
        #width = unit(10, "cm"),
        cluster_rows = FALSE,
        cluster_columns = FALSE,
        column_title = "trials",
        column_title_side = "bottom",
        #row_title = "coins",
        name = "responsibilities")
```