Sampling Cluster Indicators

Oystein Sorensen

8/16/2018

In this script I want to understand the sampling of cluster indicators in the `MCMCclustering` function, with a small simulation study. Based on the Algorithm 2 in the JMLR paper, we want to sample

$$z_{j,m} \sim \mathcal{M}(p_{1j}, \ldots, p_{C_j}).$$

In my understanding, this means performing **one** sampling, in which we draw cluster 1 with probability p_{1j}, cluster 2 with probability p_{2j}, up to cluster C with probability p_{Cj}.

Assume that $C = 3$, and that the cluster probabilities are $(0.1, 0.2, 0.7)$. In `R`, we simply draw such a sample with the following line:

```
probs <- c(0.1, 0.2, 0.7)
(z_j_m <- sample.int(n = length(probs), size = 1, prob = probs, replace = TRUE))
```

```
## [1] 3
```

Brute Force `R` Implementation

Since we do not have access to `sample.int` in `C++`, I wrote code equivalent to this function: (OK, this R code is slow, but that is not the point).

```
draw_cluster_assignments <- function(prob){

  # Draw uniform random number
  unif <- runif(1)

  # Compute the cumulative probability
  cprobs <- cumsum(probs)

  # Find the first element of the cumulative probability larger then the uniform number
  z_j_m <- min(which(cprobs > unif))

  return(z_j_m)
}
```

Let us simulate from this function a large number of times.

```
num_samples <- 10000
samples <- replicate(num_samples, draw_cluster_assignments(prob))
table(samples) / num_samples
```

```
## samples
##      1      2      3
## 0.0990 0.2026 0.6984
```

It gives exactly the probabilies I asked for.

The Implemention in MCMCclustering

This is the `C++` code that I am trying to replicate, because I do not understand it.

```
    for(i=0; i<N; i++){
      aa = updatezeta.col(i)/sum(updatezeta.col(i));
      sampleU = as_scalar(randu(1,1));
      quale = 0;
      k = 0;
      somma = 0;
      while(quale == 0){
        // aa(k) is the probability that assessor i belongs to cluster k
        //
        // When k = 0, aa(k) / (1 - somma) is the prop
        if( ( sampleU < as_scalar(aa(k))/(1-somma) ) || ( k == K-1 ) ) {
          quale = 1;
        }
        somma += as_scalar(aa(k));
        k++;
      }
      zetaold(i) = k;
    }
```

This is my `R` replication. Note that I use `k + 1` in the indexing, but otherwise it should be the same.

```
draw_cluster_assignments_cond <- function(probs){
  K <- length(probs)

  # Draw uniform random number
  sampleU <- runif(1)

  aa <- probs / sum(probs)
  quale <- 0
  k <- 0
  somma <- 0

  while(quale == 0){
    if(sampleU < aa[[k + 1]]/(1 - somma) || k == K){
      quale <- 1
    }
    somma <- somma + aa[[k + 1]]
    k <- k + 1

  }

  return(k)
}
```

This implementation does not assign clusters according to `probs`.

```
samples <- replicate(num_samples, draw_cluster_assignments_cond(probs))
table(samples) / num_samples
```

```
## samples
##      1      2      3
## 0.1018 0.1236 0.7746
```

My question is:

- Is this latter implementation correct?

Sylvia' Correction

```
draw_cluster_assignments_sylvia <- function(probs){

  K <- length(probs)

  # Draw uniform random number
  sampleU <- runif(1)
  aa <- probs / sum(probs)

  quale <- 0
  k <- 0
  somma <- 0

  while(quale == 0){
    if(sampleU < aa[[k + 1]]/(1 - somma) || k == K){
      quale <- 1
    }
    somma <- somma + aa[[k + 1]]
    k <- k + 1

    # Syliva's correction
    sampleU<-runif(1)

  }

  return(k)
}
```

This implementation seems correct!

```
samples <- replicate(num_samples, draw_cluster_assignments_sylvia(probs))
table(samples) / num_samples
```

```
## samples
##      1      2      3
## 0.1020 0.1945 0.7035
```

My C++ Implementation

I ended up writing the following `C++` code, based on the algorithm in the function `draw_cluster_assignments`. I could equally well have used Sylvia's version.

```
// Normalise the assignment probabilities, to unit L1 norm for each assessor (column)
  assignment_prob = arma::normalise(assignment_prob, 1, 0);

  for(int assessor_index = 0; assessor_index < n_assessors; ++assessor_index){
    // Draw a uniform random number
    double u = arma::randu();

    // Find the first index that is large than u. That is the cluster index.
    int cluster = arma::as_scalar(
      arma::find(arma::cumsum(assignment_prob.col(assessor_index)) > u, 1, "first"));
    cluster_indicator(t, assessor_index) = cluster;
```

```
}
```