Bayesian Statistics I

Matthieu Vignes

April 2024

Generate data from a simple genetic mixture model

We start with this small ice breaker to motivate statistical inference for genetical mixtures. No need to know about genetic (or statistical) mixtures for now.

Simulating haploid data for a single population

First, we consider simulating genotype data for `n` haploid individuals at `R` independent bi-allelic loci (positions along the genome) sampled from a population.

The term "haploid" means that each individual has only one copy of their genome. Most animals, are diploid, which means they have two copies of their genome – one inherited from the mother and the other inherited from the father. However, focusing on haploid individuals makes the ideas and code easier to follow. Once you understand the haploid case it is not too hard to extend the ideas to the diploid case. Some animals and plants have an k-ploidy, with k>2. The idea stays the same, but the combinatorial complexity and the meaning of the number of variant copies lead to a myriad of models.

The term "bi-allelic" means that the loci have two possible alleles (types), which for convenience we will label `0` and `1`.

Under these assumptions, the genotype for each individual is simply a sequence of 0s and 1s. The probability of seeing a 0 vs a 1 at each locus is determined by the "allele frequencies" at each locus, which we will specify by a vector `p`. Specifically, `p[r]` specifies the frequency of the `1` allele at locus `r`.

The following code simulates from this model.

```r
# n: number of samples
# p:  a vector of allele frequencies
r_haploid_genotypes <- function(n, p){
  R <- length(p)
  x <- matrix(nrow = n, ncol = R)
  for(i in 1:n){
    x[i,] <- rbinom(R,rep(1,R), p)
  }
  return(x)
}
```

Example

We illustrate this function by simulating a small example dataset, containing 20 individuals at 9 loci. The frequencies of the `1` allele at the loci are increasing from `0.1` at the first locus to `0.9` at the 9th locus; the pattern is not supposed to be realistic, it is just to help illustrate the idea.

As you can see (as expected, right?), the `1` allele is rarer at the earlier loci, whereas the `0` allele is rarer at the later loci.

```r
set.seed(123)
p <- seq(0.1, 0.9, length = 9)
x <- r_haploid_genotypes(20, p)
p
```

```
## [1] 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
```

```r
x
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
##  [1,]    0    0    0    1    1    1    1    0    1
##  [2,]    0    1    0    1    1    1    0    1    1
##  [3,]    0    1    1    1    1    0    1    1    1
##  [4,]    0    0    0    1    1    0    0    1    1
##  [5,]    0    0    0    0    0    1    1    1    1
##  [6,]    0    0    0    0    1    1    1    1    1
##  [7,]    0    0    0    1    1    1    1    1    1
##  [8,]    0    1    0    1    1    0    1    1    1
##  [9,]    0    0    0    0    0    0    1    1    1
## [10,]    0    0    1    0    0    0    0    0    1
## [11,]    0    0    0    1    0    1    0    1    1
## [12,]    0    0    0    0    1    1    0    0    1
## [13,]    0    0    1    0    0    0    0    1    1
## [14,]    1    0    0    1    0    1    1    1    0
## [15,]    0    0    0    1    1    0    1    1    1
## [16,]    0    1    1    1    0    1    1    1    1
## [17,]    0    0    0    0    0    0    0    1    1
## [18,]    0    0    0    0    0    1    1    1    1
## [19,]    0    0    0    0    1    1    1    1    1
## [20,]    0    1    1    1    1    1    1    0    1
```

Simulating haploid data from a mixture of two populations

Now suppose we sample from a group of individuals formed by mixing together the individuals from two different populations. This is an example of a "mixture model".

For simplicity we will assume the two different populations are mixed in equal proportions. That is the "mixture proportions" are 0.5 and 0.5. See Exerise 1 below to play with these weights. For now, notice that we require $0.5 + 0.5 = 1$: each individual **has to** be sampled from either distribution.

The following `r_simplemix` function generates data from such a model. The allele frequencies for the two populations must be specified in a matrix `P` whose first row contains the allele frequencies for population 1 and second row is the allele frequencies for population 2. (So element `P[k,r]` is the frequency of the `1` alleles in population `k` at locus `r`.) For each individual `i`, `r_simplemix` randomly samples the population (`z[i]`) to be 1 or 2, and then uses the earlier `r_haploid_genotypes` functon to generate the genotypes (`x[i,]`) from that population.

```r
# n:  number of samples
# P:  a 2xR matrix of allele frequencies
r_simplemix <- function(n,P){
  R <- ncol(P) # number of loci
  z <- rep(0,n) # used to store population of origin of each individual
  x <- matrix(nrow = n, ncol = R) #used to store genotypes

  for(i in 1:n){
    z[i] <- sample(1:2, size = 1, prob = c(0.5,0.5))
    x[i,] <- r_haploid_genotypes(1, P[z[i], ])
  }
  return(list(x=x,z=z))
}
```

Example

To illustrate this, we sample 20 individuals: in one population the frequencies are as above, whereas in the second population they are reversed. Again not at all realistic but this should help illustrate the idea.

```
set.seed(123)
P <- rbind(p,1-p)
print(P)
```

```
##   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
## p  0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9
##    0.9  0.8  0.7  0.6  0.5  0.4  0.3  0.2  0.1
```

```
sim <- r_simplemix(n = 20, P)
```

The results of the simulation are below. The first column is the population of origin and the remaining columns are the genotypes. If you look carefully, you should see that individuals from population 1 tend to have more `1` alleles in the later loci, whereas individuals from population 2 tend to have more `1` alleles in the earlier loci. This is because of the way that the allele frequencies were set up in the two populations. We repeat that of course in real data the differences between different populations will not usually show patterns like this! These fake patterns are easy to spot by eye, albeit very likely not realistic.

```
print(cbind(z = sim$z, sim$x))
```

```
##       z
##  [1,] 2 1 1 0 0 0 0 1 0 0
##  [2,] 1 0 0 0 0 1 1 1 1 0
##  [3,] 1 0 0 1 1 1 1 1 1 1
##  [4,] 1 1 0 1 0 0 0 1 1 1
##  [5,] 2 1 1 1 1 0 0 0 0 0
##  [6,] 2 1 1 1 1 0 0 1 1 0
##  [7,] 1 0 0 0 1 0 0 0 1 1
##  [8,] 1 0 0 0 0 0 1 1 1 1
##  [9,] 2 1 1 0 1 0 1 1 1 0
## [10,] 2 1 1 1 1 0 1 0 0 0
## [11,] 1 0 0 1 0 1 0 1 1 1
## [12,] 1 0 0 1 1 0 1 0 1 1
## [13,] 1 0 0 0 0 1 1 1 1 1
## [14,] 1 0 0 1 0 1 0 0 0 1
## [15,] 2 1 1 1 0 0 0 0 0 0
## [16,] 1 0 0 0 0 0 1 1 1 1
## [17,] 1 0 0 0 0 1 0 1 1 1
## [18,] 1 0 1 1 1 1 1 1 0 1
## [19,] 1 0 0 0 0 0 1 1 0 0
## [20,] 2 1 0 1 0 0 0 0 0 0
```

Inference problems

Here are some things you might like to try:

1. Modify the `r_simplemix` code to create a new funtion that allows the mixture proportions to be

specified, rather than the fixed ones (50-50). You could do this by adding a parameter `w` to the function that specifies the proportions to use (`w` for "weights").

2. Write a function, `posterior_prob_assignment=function(x, P, w)`, to compute the posterior probability that each individual came from each population, given the genotypes `x`, the allele frequencies `P` and the mixture proportions `w`. Apply your function to the simulated data.

3. Write a function `posterior_param_allele_frequencies <- function(x,z,a)` to compute the parameters of the beta posterior in each population at each locus. Here a is a vector of length 2 giving the parameters of the beta prior for P[k,r]. That is, the prior is `P[k,r] \sim Beta(a[1],a[2])`. Because the Beta distribution has two parameters, there will be 2 parameters for each locus and each population. So the output of your function should be a `2 x K x R` array (where `K=2` because we have a mixture of 2 populations).