Generate data from a simple genetic mixture model

Matthew Stephens *University of Chicago*

See here for a PDF version of this vignette.

Introduction

This vignette is an "ice breaker" to motivate learning and statistical inference centered around genetic mixtures.

You don't need to know what a mixture model is to understand this. But if you want to know more about mixture models in general you might read this introduction to mixture models.

Simulating haploid data for a single population

First we consider simulating genotype data for n haploid individuals at R independent biallelic 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, focussing 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.)

The term "biallelic" 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 zeros and ones. 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.

```
#' @param n The number of samples.
#' @param p A vector of allele frequencies for R loci.
#' @return An n x R matrix of haploid genotypes.
r_haploid_genotypes <- function (n, p) {
  R <- length(p)
  x <- matrix(0,n,R)
  for (i in 1:n)
    x[i,] <- rbinom(R,rep(1,R),p)
  return(x)
}
```

Example

To illustrate this function, we simulate a small example dataset containing 20 haploid 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 ninth locus. (The pattern is not supposed to be realistic, it is just to help illustrate the idea.)

```
set.seed(123)
p <- seq(0.1,0.9,length.out = 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
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
```

As you can see, the 1 allele is rarer at the earlier loci, whereas the 0 allele is rarer at the later loci.

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.

The following 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_{kr} is the frequency of the 1 alleles in population k at locus r.) For each individual i, it randomly samples

the population $z_i = k$ to be 1 or 2, and then uses the `r_haploid_genotypes` function to generate the genotypes from that population.

```
#' @param n The number of samples.
#' @param P A 2 x R matrix of allele frequencies.
#' @return A list containing a matrix of genotypes (x) and
#'   the populations of origin (z).
r_simplemix <- function (n, P) {
  R <- ncol(P)
  z <- rep(0,n)
  x <- matrix(0,n,R)
  for (i in 1:n) {
    z[i]  <- sample(2,1,prob = c(0.5,0.5))
    k     <- z[i]
    x[i,] <- r_haploid_genotypes(1,P[k,])
  }
  return(list(x = x,z = z))
}
```

Example

In this example, we sample 20 individuals: in one population, the frequencies are as above, whereas in the second population they are reversed $(1 - p)$. Again, this is not at all realistic, but intended to illustrate the ideas.

```
set.seed(123)
P <- rbind(p,1 - p,deparse.level = 0)
print(P)
#      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
# [1,]  0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9
# [2,]  0.9  0.8  0.7  0.6  0.5  0.4  0.3  0.2  0.1
sim <- r_simplemix(20,P)
```

Here are the results of the simulation:

```
colnames(sim$x) <- paste0("r",1:9)
cbind(z = sim$z,sim$x)
#       z r1 r2 r3 r4 r5 r6 r7 r8 r9
#  [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
```

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. Of course, in real data the differences between different populations will not usually show patterns like this! I chose the patterns so you can see them by eye.

Inference problems

These types of data can help motivate a number of statistical inference problems, including:

1. Given the allele frequencies **P** and the genotypes **X**, how might you infer the populations of origin z? In the genetic literature, this is sometimes called the "assignment problem"; more generally, it is an example of a "classification problem".
2. Given the populations of origin z and the genotypes **X**, how might you infer the population allele frequencies **P**?
3. Given just the genotypes **X** how might you infer *both* z and **P**? This is an example of a "clustering problem".

Exercises

Here are some things you might like to try:

1. Modify the `r_simplemix` function to allow the mixture proportions to be specified as an input argument, rather than fixed at $(0.5, 0.5)$. You could do this by adding a parameter "w" to the function that specifies the proportions to use ("w" is short for "weights").
2. Write a function `posterior_prob_assignment(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 data you simulated.
3. Write a function `posterior_param_allele_frequencies(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_{kr}; that is, the prior is $p_{kr} \sim \text{Beta}(a_1, a_2)$. Because the Beta distribution has two parameters, there will be 2 parameters for each locus and for each population. So the output of your function should be two $2 \times R$ matrices, or a $2 \times 2 \times R$

array.

Answer templates

Here are some answer templates for Exercises 2 and 3. You will need to fill in the "add your code here" comments. (*Note:* some of these code chunks have the option set `eval = FALSE` so that this document compiles. You will have to remove this option if you want these chunks to run on compilation.)

Exercise 2

Function to compute the likelihood:

```
#' @param p A vector of allele frequencies of length R.
#' @param x A vector of genotypes of length R.
#' @return The likelihood for the allele frequences (p) given the
#'   genotype data (x).
likelihood <- function (p, x)
  prod(p^x*(1-p)^(1-x))
```

Function to "normalize" a vector:

```
#' @param x A vector.
#' @return The normalized vector in which its elements sum to 1.
normalize <- function (x)
  x/sum(x)
```

Function to compute the posterior assignment probabilities:

```
#' @param x An n x R matrix of genotypes.
#' @param P A 2 x R matrix of allele frequencies.
#' @param w A vector of mixture proportions of length 2.
#' @return An n x 2 matrix of posterior assignment probabilities.
posterior_prob_assignment <- function (x, P, w) {
  K <- 2
  n <- nrow(x)
  post <- matrix(0,n,K)
  lik <- rep(0,K)
  for (i in 1:n) {
    for (k in 1:K) {
      # Add your code here.
    }
    # Add your code here.
  }
  return(post)
}
```

Once you have implemented this function, try running it on an example data set, and compare the posterior assignment probabilities to the true labels:

```
set.seed(123)
sim <- r_simplemix(20,P)
posterior <- posterior_prob_assignment(sim$x,P,w = c(0.5,0.5))
plot(sim$z,posterior[,2],xlab = "true population",
     ylab = "posterior prob for population 2")
```

Exercise 3

Function to compute the posterior distribution of the allele frequencies for each population and for each locus.

```
#' @param x An n by R matrix of genotypes.
#' @param z An vector of population assignments of length n.
#' @param a A vector of length 2 specifying the Beta prior.
#' @return A 2 x 2 x R array containing the posterior parameters for the
#'    the allele frequencies given the genotypes and population labels.
posterior_param_allele_frequencies <- function (x, z, a) {
 K <- 2
 R <- ncol(x)
 post_param <- array(0,c(2,K,R))
 for (k in 1:K) {
   for (r in 1:R) {
     # Add your code here.
   }
 }
 return(post_param)
}
```

Once you have implemented this function, try running it on an example data set

```
set.seed(123)
sim <- r_simplemix(20,P)
posterior_param <- posterior_param_allele_frequencies(sim$x,sim$z,a = c(1,1))
posterior_param[,1,] # Posterior for population 1.
posterior_param[,2,] # Posterior for population 2.
```