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 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, 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 "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
#' @param n number of samples
#' @param 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

To illustrate this function we simulate 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, the `1` allele is rarer at the earlier loci, whereas the `0` allele is rarer at the later loci.

```
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
```

```
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.

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` it randomly samples the population (`z[i]`) to be 1 or 2, and then uses the `r_haploid_genotypes` to generate the genotypes (`x[i,]`) from that population.

```
#' @param n number of samples
#' @param P a 2 by R 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

Here we sample 20 individuals: in one population the frequencies are as above, whereas in the second population they are reversed (`1-p`)...again not at all realistic but to help illustrate an 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)
```

Here are the results of the simulation (first column is the population of origin; 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. (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, not to be 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

These types of data can help motivate a number of statistical inference problems.

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` code to allow the mixture proportions to be specified, 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` 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 parame-

ters 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).

Answer templates

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

Exercise 2

```r
#' Compute the likelihood for allele frequences p given genotype data x
#' @param p an R vector of allele frequencies
#' @param x an R vector of genotypes
likelihood = function(p,x){
  return(prod(p^x*(1-p)^(1-x)))
}

#' normalize a vector, x, so it's elements sum to 1
#' @param x a vector
normalize = function(x){x/sum(x)}

#' Compute posterior assignment probabilities
#' @param x an n by R matrix of genotypes
#' @param P a 2 by R matrix of allele frequencies
#' @param w a 2-vector of prior probabilities
posterior_prob_assignment=function(x,P,w){
  n = nrow(x)
  K = length(w)
  post = matrix(nrow=n, ncol=K) # to store the posterior probabilities
  lik = rep(0,K) # a vector to store the likelihoods
  for(i in 1:n){
    for(k in 1:K){
    ...
    }
    ...
  }
  return(post)
}
```

Here we can run this on the example data, and compare the posterior probabilities with the true labels.

```r
set.seed(123)
sim = r_simplemix(n=20,P)
```

```r
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

```r
#' Compute the posterior parameters for allele frequencies given genotypes
#' and population labels
#' @param x an n by R matrix of genotypes
#' @param z an n vector of population assignments
#' @param a a 2-vector of prior parameters
posterior_param_allele_frequencies=function(x,z,a){
 K = 2
 R = ncol(x)
 post_param = array(dim=c(2,K,R))
 for(k in 1:K){
   for(r in 1:R){
     ...
   }
 }
 return(post_param)
}
```

Here we can run this on the example data:

```r
set.seed(123)
sim = r_simplemix(n=20,P)
posterior_param = posterior_param_allele_frequencies(sim$x,sim$z,a = c(1,1))
posterior_param[,1,] # these should be the posterior parameters for population 1
posterior_param[,2,] # these should be the posterior parameters for population 2
```