COMP 5970/6970-004
Computational Biology: Genomics and Transcriptomics
Lecture notes 5: 1/27/2022

Haynes Heaton

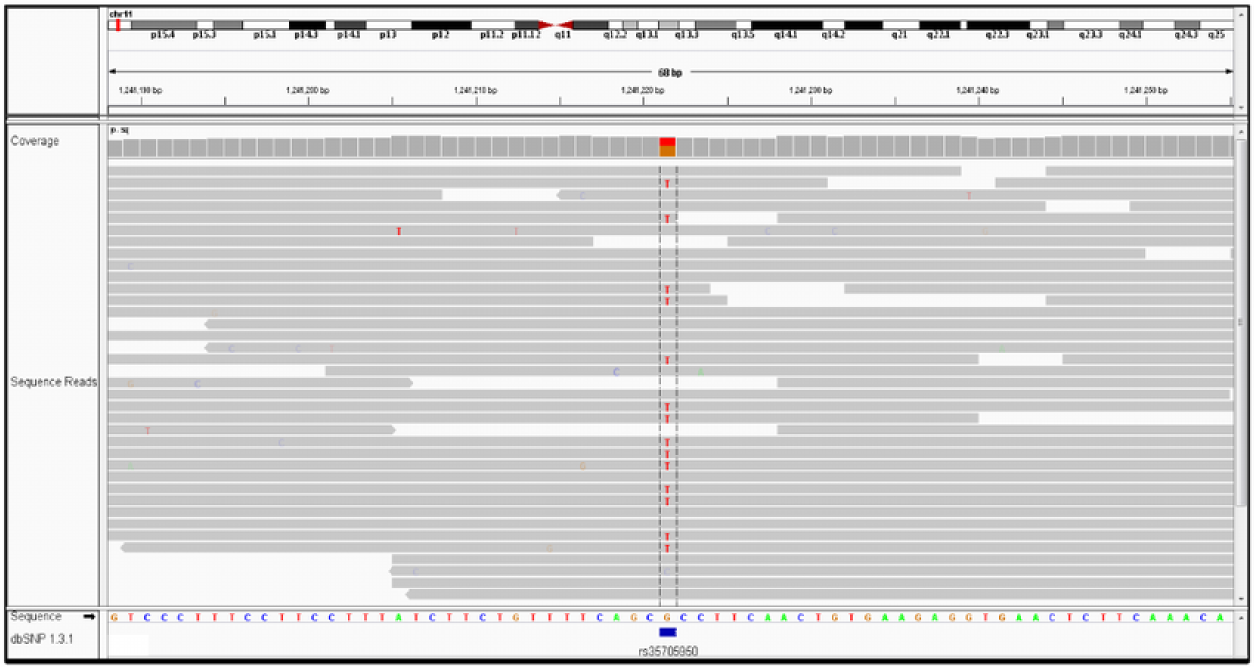
Spring, 2022

Lecture Objectives

- Statistical models
  - Compound distributions
  - Mixture models
- Statistical inference
  - Maximum likelihood estimation (MLE)
  - Expectation Maximization (EM)

Statistical models

A statistical model is simply a mathematical explanation of how data is generated. This may be a single simple distribution, or it may be a combination of distributions. A poisson distribution may produce the n parameter for a binomial distribution, for example. In short read DNA sequencing, we generate millions of reads randomly from anywhere in the genome. The coverage, or depth, of reads from any one location is most practically modeled with a Poisson distribution as the support for this number is $[0, millions]$. Technically, this could be modeled with a binomial with $n = millions$ and a very very small p, but this leads to computational problems when calculating $\binom{n}{k}$ with large n. Then within these reads, there can be genetic variants. We have two copies of every chromosome, one from our mother and one from our father. When these are different, the variant is said to be **heterozygous**. Because each read randomly comes from one chromosome or the other, the number of alternative alleles (T in the example below) is binomial distributed with p of 0.5 and n as the depth that was sampled from a Poisson distribution. This is an example of a compound distribution.



Another type of compound distribution is when each data point may be generated by one of multiple different distribution. This is called a mixture model. The marginal likelihood of a mixture model looks like the following.

$$p(D|dist_1)p(dist_1) + p(D|dist_2)p(dist_2) \tag{1}$$

From the same example above, we can construct a mixture model. At every base, the genome may be all reference alleles with some errors, all alternative alleles with some errors, or heterozygous (with some errors). For the purposes of this example, let us imagine that that there are only 2 options for any location—or just count all 3 other base options as errors. In the data shown, the error rate is very low, but there are other DNA sequencing technologies with higher error rates. What if we want to estimate this error rate? We could construct a model of this mixture of distributions with unknown parameters that capture the error rate. Each of these 3 distributions are binomials with different probability parameters. For each location, k will be the number of alternative bases observed from these binomials. The marginal likelihood will be the following

$$\mathcal{L} = p(D|\text{binom}, p_1)p(\text{binom}, p_1) + p(D|\text{binom}, p_2)p(\text{binom}, p_2) + p(D|\text{binom}, p_3)p(\text{binom}, p_3) \tag{2}$$

For simplicity, we can just have equal priors. So our goal is to learn $p = [p_1, p_2, p_3]$. We will use a **maximum likelihood estimate** approach.

$$\underset{p}{\text{argmax}}(\mathcal{L}) \tag{3}$$

1 Expectation Maximization

To do this, we will use **Expectation Maximization (EM)**. This works by starting with random initializations of p_1, p_2, and p_3, obtaining the posterior probability that each data point arose from each distribution, and then updating the probability parameter of each distribution according to the data weighted on the posterior probabilities that each data point arose from that distribution. You can intuitively think of this as each data point pulling each distribution toward itself weighted on how likely it arose from that distribution. The first step of finding the posterior probability that each data point came from each distribution is called the **expectation** step. Updating the distributions according to which data points posterior probability preferred them is the **maximization** step. We alternate these two steps until the parameters converge to a solution and they change very little with further steps. The posteriors can be computed with bayes theorem.

$$p(\text{binom}, p_i|D) = \frac{p(D|\text{binom}, p_i)p(\text{binom}, p_i)}{p(D|\text{binom}, p_1)p(\text{binom}, p_1) + p(D|\text{binom}, p_2)p(\text{binom}, p_2) + p(D|\text{binom}, p_3)p(\text{binom}, p_3)} \tag{4}$$

This is generally done in log space for numerical stability reasons. In the floating point representation in computers, there is a limited number of possible values extremely near 0, but the log of a probability very near 0 has much more dynamic range in the floating point representation. The one wrinkle to this comes in the denominator.

$$\begin{aligned}
&log(p(\text{binom}, p_i|D)) = \\
&\qquad log(p(D|\text{binom}, p_i)) + log(p(\text{binom}, p_i)) - \\
&\qquad \text{logsumexp}(\\
&\qquad\qquad log(p(D|\text{binom}, p_1)) + log(p(\text{binom}, p_1)), \\
&\qquad\qquad log(p(D|\text{binom}, p_2)) + log(p(\text{binom}, p_2)), \\
&\qquad\qquad log(p(D|\text{binom}, p_3)) + log(p(\text{binom}, p_3)) \\
&\qquad\qquad)
\end{aligned} \tag{5}$$

The logsumexp function takes the log of the sum of e^x as it says and can be implemented in a numerically stable way. Make sure to use packages that have implemented this function and if you need to implement it yourself, look up how to do so to keep in numerically stable.

And the maximization step will be

$$\hat{p}_i = \frac{\sum_j p(\text{binom}, p_i|D_j)k_j}{\sum_j p(\text{binom}, p_i|D_j)n_j} \tag{6}$$

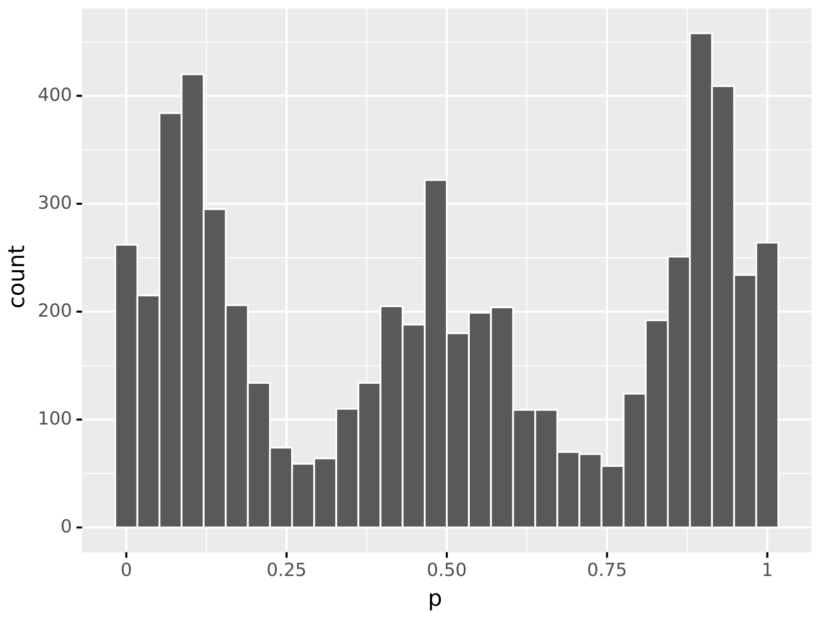
where k_j and n_j are the k and n of the jth data point.

```
## /Library/Frameworks/R.framework/Versions/4.1/Resources/library/reticulate/python/rpytools/loader.py:3
```

Let's generate some sample data and plot it.

```
samples = 2000

ns1 = np.random.poisson(20,samples)
ns2 = np.random.poisson(20,samples)
ns3 = np.random.poisson(20,samples)

k1 = np.random.binomial(ns1, 0.5, samples)
k2 = np.random.binomial(ns2, 0.1, samples)
k3 = np.random.binomial(ns3, 0.9, samples)

ns = []
ns.extend(ns1)
ns.extend(ns2)
ns.extend(ns3)
ks = []
ks.extend(k1)
ks.extend(k2)
ks.extend(k3)
```

```
p = np.asarray(ks) / np.asarray(ns)
dataframe = pd.DataFrame(data={'n':ns, 'k':ks, 'p':p})

ggplot(dataframe)+geom_histogram(aes(x='p'),color="white",bins = 30)

## <ggplot: (8784509357650)>
```



Now let's randomly initialize the probabilities.

```
ps = np.random.uniform(size=3)
ps

## array([0.89653981, 0.41466995, 0.67324919])
```

And set the log priors to be uniform.

```
log_priors = [np.log(1/3)]*3
log_priors

## [-1.0986122886681098, -1.0986122886681098, -1.0986122886681098]
```

Now we can do the expectation step.

```
def calc_posteriors(n,k,ps,log_priors):
    log_likelihoods = scipy.stats.binom.logpmf(k,n,ps)
    log_posteriors = []
    log_marginal_likelihoods = log_likelihoods + log_priors
    for (i, log_likelihood) in enumerate(log_likelihoods):
        log_posteriors.append(log_likelihood + log_priors[i] -
            scipy.special.logsumexp(log_marginal_likelihoods))
    return np.exp(log_posteriors)
```

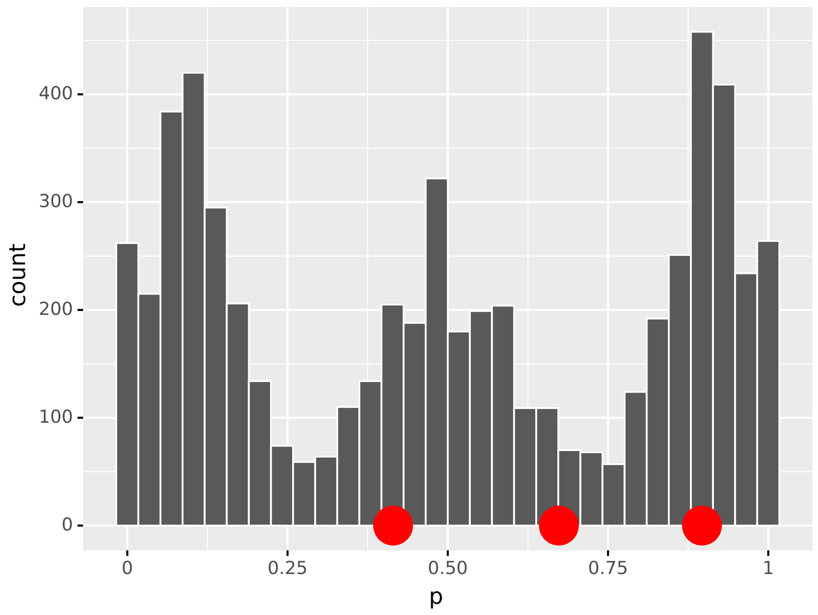
```
def expectation(data, ps):
    all_posteriors = []
    for i in range(len(data)):
        posteriors = calc_posteriors(data['n'][i], data['k'][i], ps, log_priors)
        all_posteriors.append(posteriors)
    return all_posteriors
```

And the maximization step

```
def maximization(all_posteriors, data, ps):
    k_primes = [0]*3
    n_primes = [0]*3
    for i in range(len(all_posteriors)):
        for (j, posterior) in enumerate(all_posteriors[i]):
            k_primes[j] += posterior * data['k'][i]
            n_primes[j] += posterior * data['n'][i]

    p_primes = [k_primes[i]/n_primes[i] for i in range(3)]
    return p_primes
```

Now let's plot the probability parameters on the data before and after running expectation maximization.

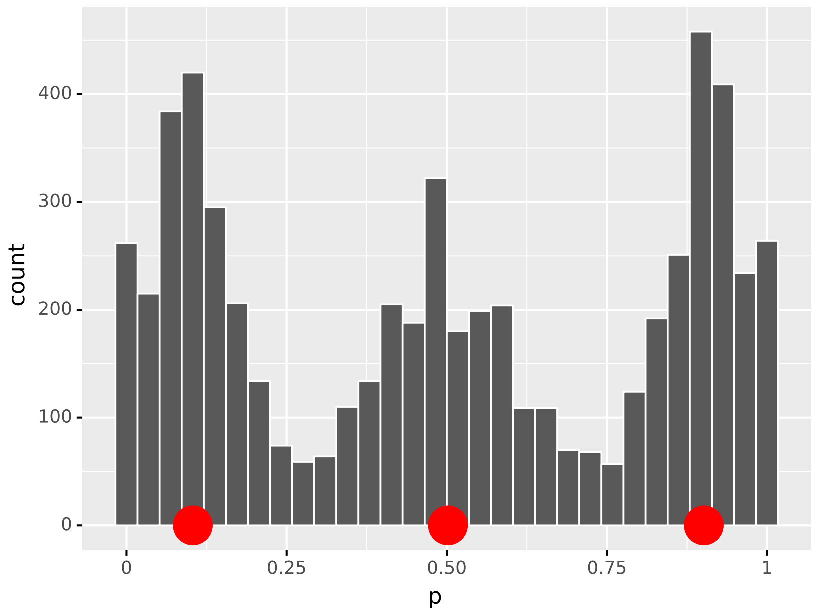
```
p_data = pd.DataFrame(data={'p':ps,'y':[0]*3})
(ggplot(dataframe)+geom_histogram(aes(x='p'),color="white",bins = 30)+
  geom_point(aes(x='p',y='y'), data = p_data ,color="red", size = 10))

## <ggplot: (8784509384500)>
```



Run EM.

```
for i in range(6):
  all_posteriors = expectation(dataframe, ps)
  ps = maximization(all_posteriors, dataframe, ps)
```

```
p_data = pd.DataFrame(data={'p':ps,'y':[0]*3})
(ggplot(dataframe)+geom_histogram(aes(x='p'),color="white",bins = 30)+
  geom_point(aes(x='p',y='y'), data = p_data ,color="red", size = 10))

## <ggplot: (8784525922497)>
```



As you can see, we have now learned the underlying probabilities from our mixture model.