COMP 5970/6970-004
Computational Biology: Genomics and Transcriptomics
Lecture notes 10: 2/15/2022

Haynes Heaton

Spring, 2022

Lecture Objectives

- More on mixture models - clustering

Mixture model clustering

So we have been using mixture models and finding their optimal parameters with maximum likelihood estimation and expectation maximization. But I have not covered maybe the most practical application of mixture models which is clustering data. When there are multiple clusters in data, it can be modeled as a mixture of generating distributions. When we do expectation maximization, when the parameters have converged, the expectation step is simply the posterior probability of each data point coming from each model—or the probability of each data point coming from each cluster!

$$p(M_0|D_i) = \frac{p(D_i|M_0)p(M_0)}{\sum_j (p(D_i|M_j)p(M_j))} \tag{1}$$

This is most frequently done with a mixture of gaussians in what is known as a **gaussian mixture model or GMM**. But this clustering can be applied to any type of distribution. In the example of estimating the error rate of DNA sequencing reads, we did had a mixture of binomials. And while we focused on the estimation of the distribution parameters, we were also implicitly clustering the individual data points. I will now demonstrate this process with a GMM.

```
import numpy as np
import pandas as pd
import scipy
from plotnine import *

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

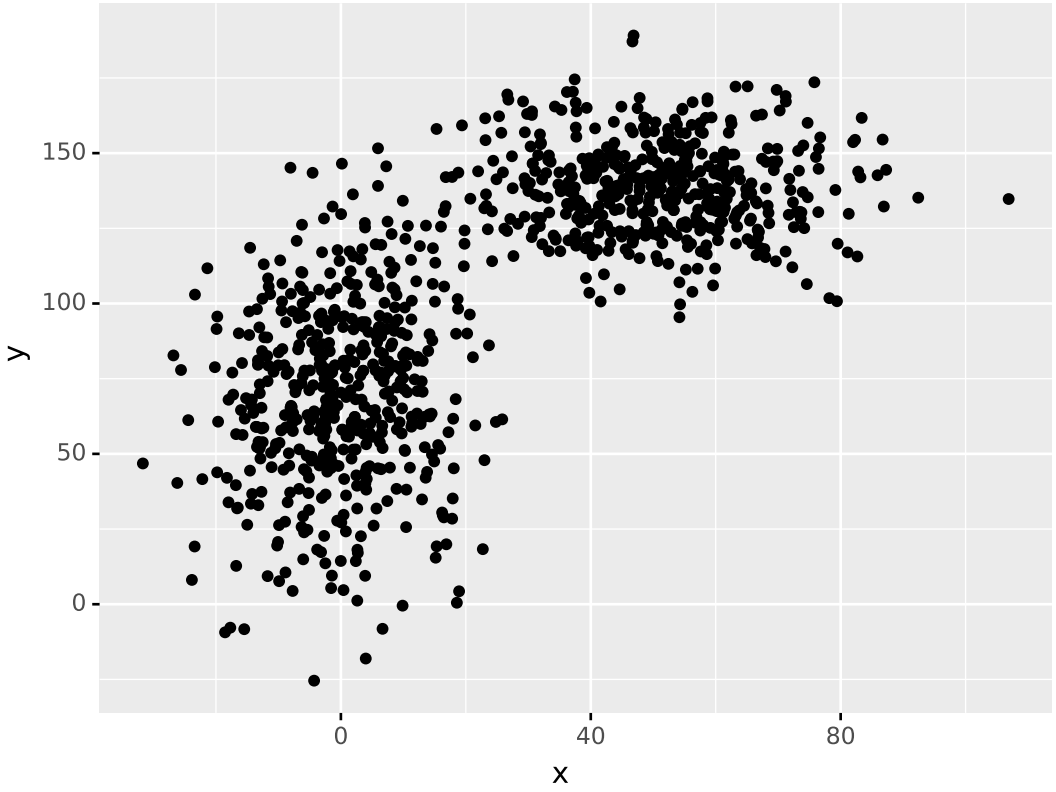
First lets generate some data to cluster.

```
mus1 = [50,140]
mus2 = [0,70]
sds1 = [15,15]
sds2 = [10,30]
samples = 500
```

```
x1 = np.random.normal(mus1[0], sds1[0], samples)
y1 = np.random.normal(mus1[1], sds1[1], samples)
x2 = np.random.normal(mus2[0], sds2[0], samples)
y2 = np.random.normal(mus2[1], sds2[1], samples)
df = pd.DataFrame({'x':np.append(x1,x2), 'y':np.append(y1,y2)})
```

And we can plot this data.

```
ggplot(df)+geom_point(aes(x='x',y='y'))

## <ggplot: (8759497173468)>
```

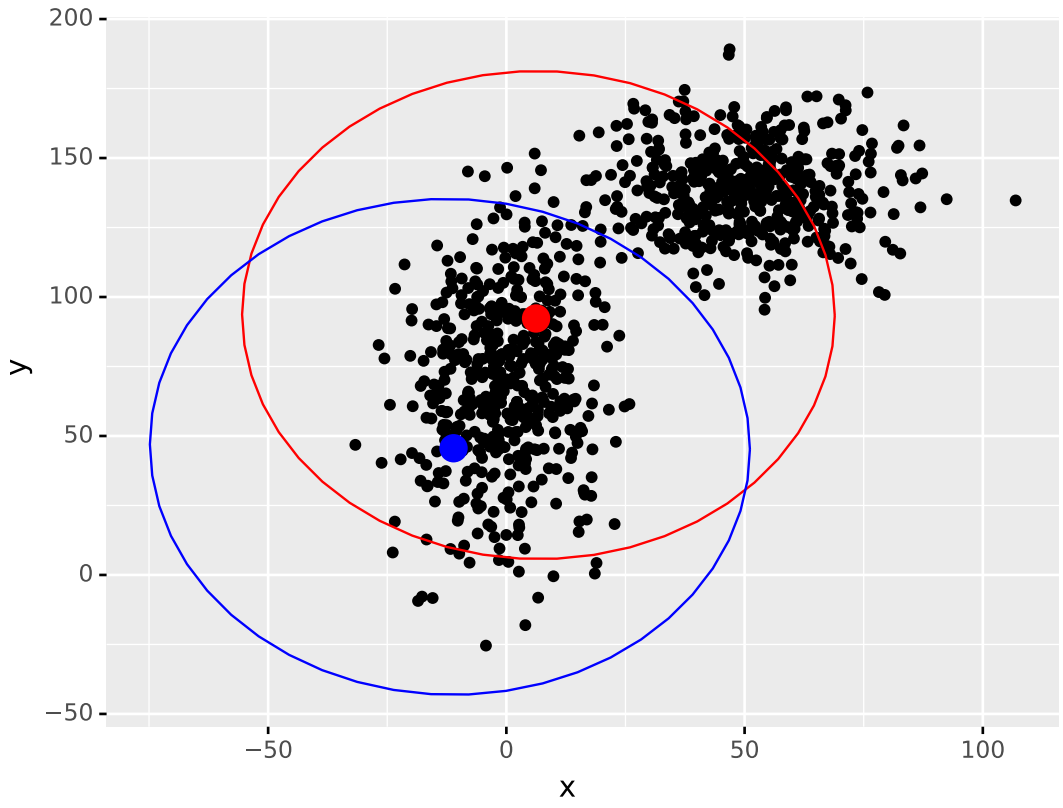


Now we want to initialize our starting parameters. For the means, it is common to choose data points from the data set for the initial values. For the standard deviations, we want values that are not too big and not too small at least by orders of magnitude. Given the plot above, I went with the standard deviation of the whole data as a starting point and over iterations, the EM will learn the correct standard deviations.

```
mus1_index = np.random.choice([i for i in range(len(x1)+len(x2))])
mus2_index = np.random.choice([i for i in range(len(x1)+len(x2))])
mus1 = [df['x'][mus1_index], df['y'][mus1_index]]
mus2 = [df['x'][mus2_index], df['y'][mus2_index]]
sds1 = [np.std(df['x']), np.std(df['y'])]
sds2 = [np.std(df['x']), np.std(df['y'])]
mus = [mus1,mus2]
sds = [sds1,sds2]
```

Now we can plot where these centers are and an envelope representing the standard deviations. There is a geom_density_2d but it is currently not working in plotnine. I have submitted a bug report. But there is a stat_ellipse which, given some data, finds the 95th percentile envelope so I'll use that for now. In order to get this envelope, I need to simulate samples from the cluster distributions first.

```
x1_samp = np.random.normal(mus1[0],sds1[0],1000)
y1_samp = np.random.normal(mus1[1],sds1[1],1000)
x2_samp = np.random.normal(mus2[0],sds2[0],1000)
y2_samp = np.random.normal(mus2[1],sds2[1],1000)
df3 = pd.DataFrame({'x':x1_samp,'y':y1_samp})
df4 = pd.DataFrame({'x':x2_samp,'y':y2_samp})

df1 = pd.DataFrame({'x':[mus1[0]],'y':[mus1[1]]})
df2 = pd.DataFrame({'x':[mus2[0]],'y':[mus2[1]]})
(ggplot(df)+geom_point(aes(x='x',y='y'))+
 geom_point(data=df1, mapping=aes(x='x',y='y'),color='red',size=5)+
   geom_point(data=df2, mapping=aes(x='x',y='y'),color='blue',size=5)+
    stat_ellipse(data=df3,mapping=aes(x='x',y='y'),color='red')+
    stat_ellipse(data=df4,mapping=aes(x='x',y='y'),color='blue'))

## <ggplot: (8759477101159)>
```



And we need priors for each distribution. We will calculate in log space.

```
log_priors = np.zeros(2)
log_priors[0] = np.log(0.5)
log_priors[1] = np.log(0.5)
```

Now for our expectation step. In the line with log_posts, I'm using broadcasting to get a $2x1$ from a $2x1$, $2x1$, and $1x1$ (the logsumexp).

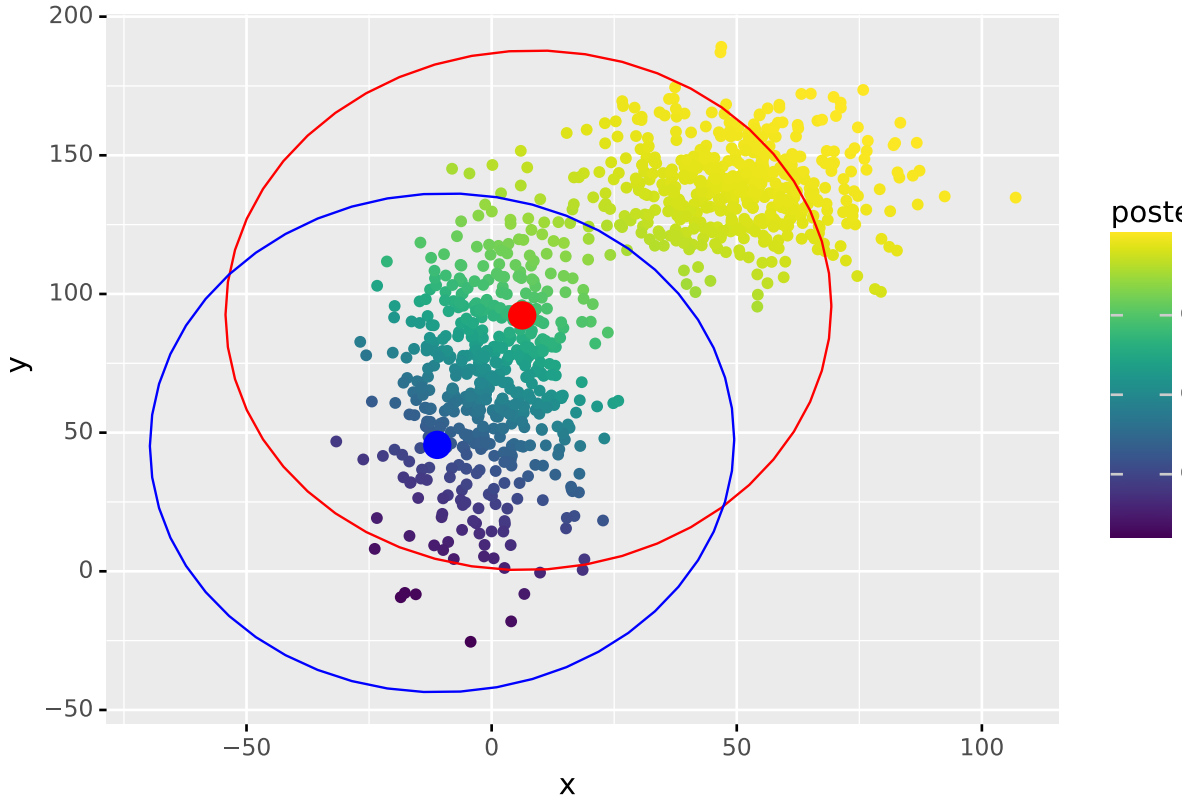
```
def expectation(mus, sds, data, log_priors):
    posteriors = []
    for i in range(data.shape[0]):
        x = data['x'][i]
        y = data['y'][i]
        log_likelihoods = np.zeros(len(mus))
        for distribution in range(len(mus)):
            log_likelihoods[distribution] = (
              calc_log_likelihood(x, y, mus[distribution], sds[distribution]))

        log_posts = (log_likelihoods + log_priors -
            scipy.special.logsumexp(log_likelihoods + log_priors))
        posteriors.append(np.exp(log_posts))
    return(posteriors)
```

```
def calc_log_likelihood(x,y,mus,sds):
    logpdf_x = scipy.stats.norm.logpdf(x, mus[0], sds[0])
    logpdf_y = scipy.stats.norm.logpdf(y, mus[1], sds[1])
    return(logpdf_x + logpdf_y)
```

Let's run expectation and then plot our data with the posteriors for cluster 0 as the color.

```
posteriors = expectation(mus, sds, df, log_priors)
```

```
x1_samp = np.random.normal(mus[0][0], sds[0][0],1000)
y1_samp = np.random.normal(mus[0][1], sds[0][1],1000)
x2_samp = np.random.normal(mus[1][0], sds[1][0],1000)
y2_samp = np.random.normal(mus[1][1], sds[1][1],1000)
df3 = pd.DataFrame({'x':x1_samp,'y':y1_samp})
df4 = pd.DataFrame({'x':x2_samp,'y':y2_samp})
df['posteriors'] = np.asarray(posteriors).T[0]
df1 = pd.DataFrame({'x':[mus[0][0]], 'y':[mus[0][1]]})
df2 = pd.DataFrame({'x':[mus[1][0]], 'y':[mus[1][1]]})
(ggplot(df)+geom_point(aes(x='x', y='y', color='posteriors'))+
 geom_point(data=df1, mapping=aes(x='x', y='y'), color='red', size=5)+
   geom_point(data=df2, mapping=aes(x='x', y='y'),color='blue', size=5)+
    stat_ellipse(data=df3,mapping=aes(x='x', y='y'), color='red')+
    stat_ellipse(data=df4,mapping=aes(x='x', y='y'), color='blue'))

## <ggplot: (8759444112354)>
```



Despite the initial cluster centers being from the same cluster, the data will push them apart to better explain the data as a whole.

Now for the maximization step. We keep track of numerators and denominators for both the means and the variances. We do variances and not standard deviations as variances are additive whereas standard deviations are not. An easy way of thinking about why this is is if you consider each data point adding to the variation in an independent way, you can think of two data points as the two short sides of a triangle and the hypotenuse as the total variation. So $a^2 + b^2 = c^2$. We must go through the data twice, first to estimate the means and from the means and the data we can estimate the variances.

```
def maximization(posteriors, data):
    mus_numerator = np.zeros((2,2))
    # x and y for each distribution. this is distribution by [x,y]
    mus_denominator = np.zeros((2,2))
    variance_numerator = np.zeros((2,2))
    variance_denominator = np.zeros((2,2))
    for i in range(data.shape[0]):
        x = data['x'][i]
        y = data['y'][i]
        for distribution in range(len(posteriors[i])):
            mus_numerator[distribution][0] += posteriors[i][distribution]*x
            mus_numerator[distribution][1] += posteriors[i][distribution]*y
            mus_denominator[distribution][0] += posteriors[i][distribution]
            mus_denominator[distribution][1] += posteriors[i][distribution]
    mus = mus_numerator / mus_denominator
```

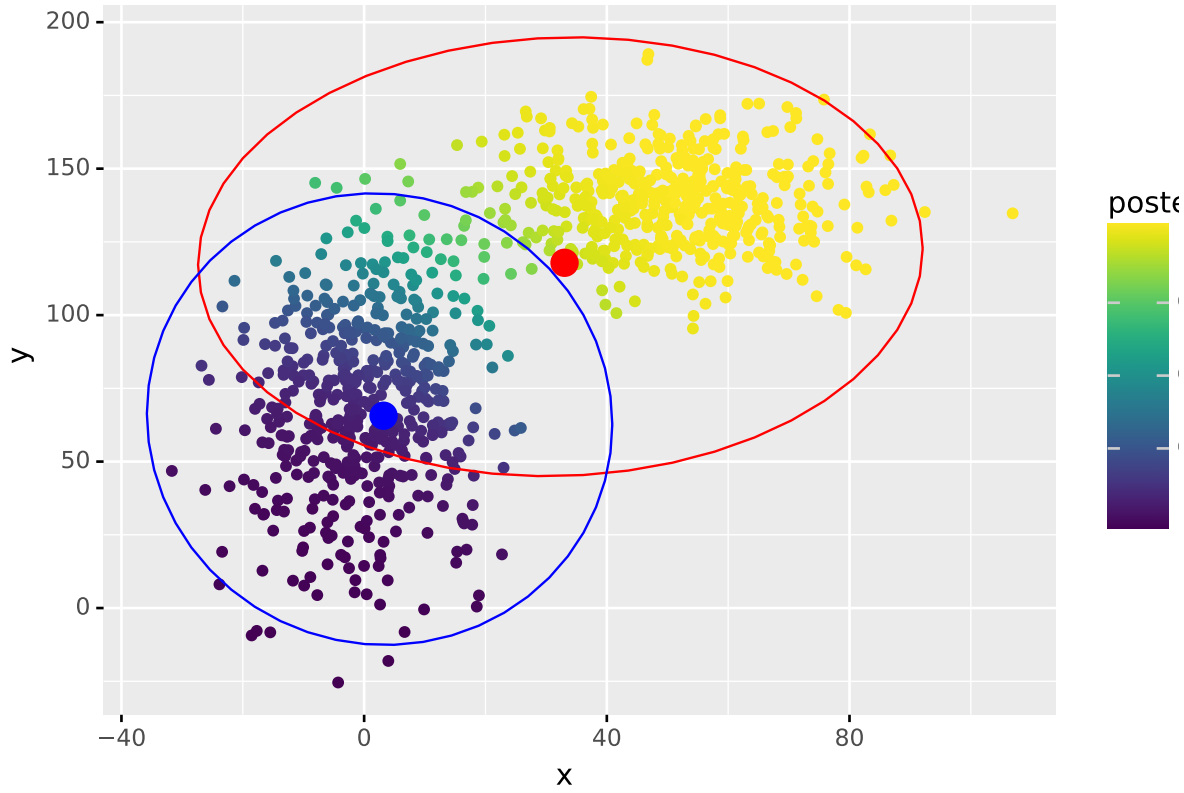
```
    for i in range(data.shape[0]):
        x = data['x'][i]
        y = data['y'][i]
        for distribution in range(len(posteriors[i])):
            variance_numerator[distribution][0] += posteriors[i][distribution]*(mus[distribution][0] - x
            variance_numerator[distribution][1] += posteriors[i][distribution]*(mus[distribution][1] - y
            variance_denominator[distribution][0] += posteriors[i][distribution]
            variance_denominator[distribution][1] += posteriors[i][distribution]

    sds = np.sqrt(variance_numerator / variance_denominator)
    return(mus, sds)
```

Now lets run maximization and plot again.

```
(mus, sds) = maximization(posteriors, df)
posteriors = expectation(mus, sds, df, log_priors)
```

We now see the clusters slowly moving in the right direction.

```
## <ggplot: (8759444113264)>
```



And we can run EM a few more steps and plot the final output.

```
for i in range(6):
  posteriors = expectation(mus, sds, df, log_priors)
  (mus, sds) = maximization(posteriors, df)
```

```
## <ggplot: (8759468391632)>
```