A worked-out example for EM

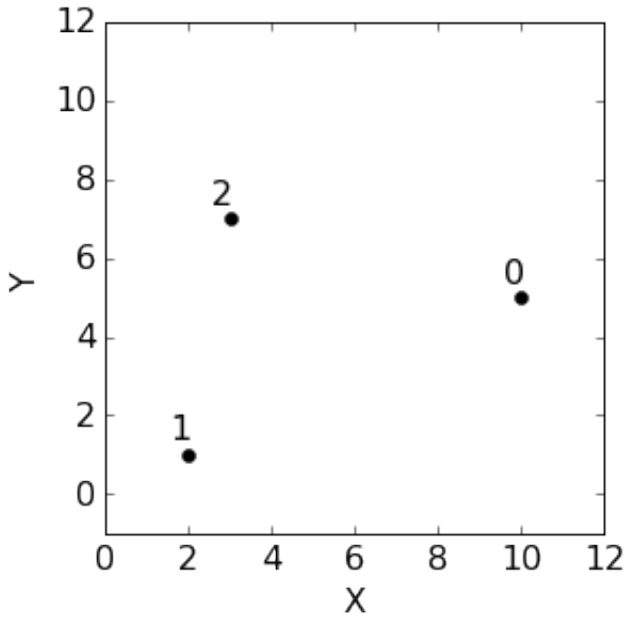
When it comes to k-means, the notion of clusters is straightforward — each point belongs to the cluster with the nearest centroid (mean). If someone gave us some data points and centroids, we can readily label each point for cluster.

The notion of clusters for EM is not as immediately intuitive. Before jumping into the minutiae of EM, it is best to develop your intuition as to what clusters really mean.

Prelude: cluster assignment in k-means

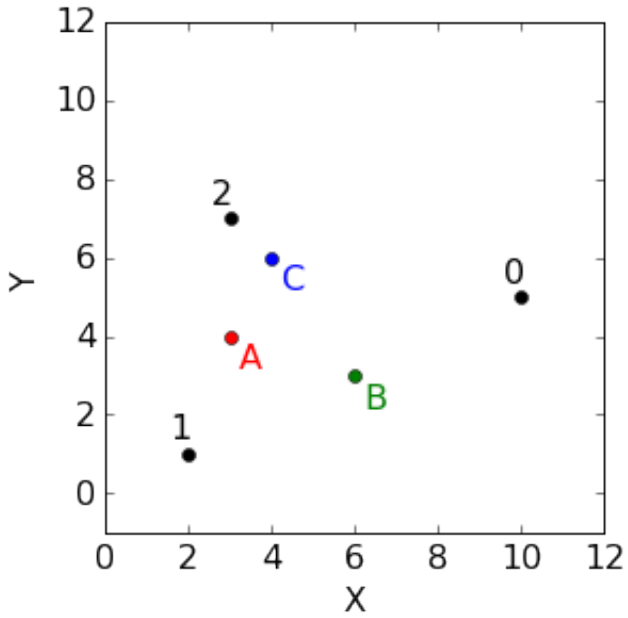
Let us consider a toy example with three data points in 2D:

Dataset	X	Y
Data point 0	10	5
Data point 1	2	1
Data point 2	3	7



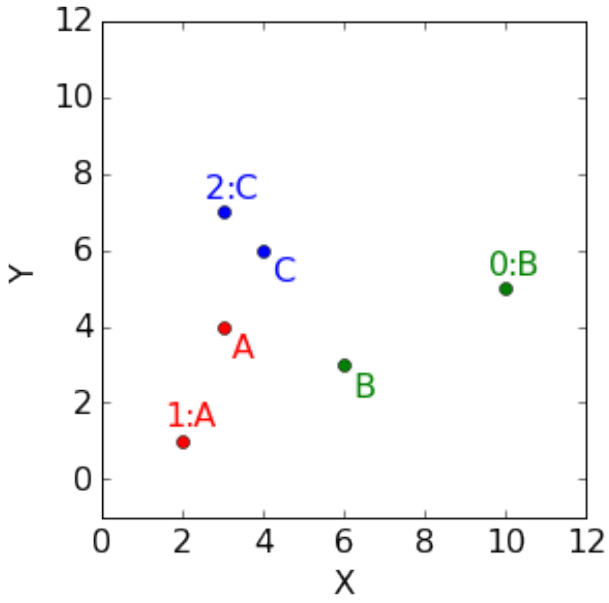
If we were running k-means, we would first take an initial set of centroids and then label all points with the closest ones. Let's take the following centroids:

Centroids	X	Y
Cluster A	3	4
Cluster B	6	3
Cluster C	4	6



(Since this is a toy example, let's ignore the fact that having three clusters over three data points is an overkill. But I digress.) Using the pairwise Euclidean distances, we label the data points with nearest clusters:

Pairwise	Centroid of	Centroid of	Centroid of	**Cluster**
Data point 0	7.071	**4.472**	6.083	**Cluster B**
Data point 1	**3.162**	4.472	5.385	**Cluster A**
Data point 2	3.000	5.000	**1.414**	**Cluster C**



So far so good. But what does it mean for a data point to be assigned to a cluster? For instance, data point 1 is "assigned" to cluster A and none other. That is, the entire fraction (100%) of data point 1 is given to cluster A and zero fraction to cluster B and cluster C. Following the same reasoning, we can write out the fractions of all data points going to all clusters:

	Cluster A	Cluster B	Cluster C
Data point 0	0	**1**	0
Data point 1	**1**	0	0
Data point 2	0	0	**1**
Member counts	1	1	1

Notice that exactly one cluster gets a 1 for each row, because fractions of each point must add up to 1, e.g. 100%. Also, note that each column sums to the number of data points in the corresponding cluster. All column sums happened to be 1 in this particular example, as each cluster gets exactly one point. In general:

- Summing over a row always yields 1, as all fractions of a point must add up to a whole.
- Summing over a column yields the number of data points assigned to the corresponding cluster.

Cluster assignment in EM

So far, we've been insisting that each data point be exclusively assigned to a single cluster and no other. This often proves to be limiting, particularly when it is useful to express uncertainties regarding its cluster membership.

Looking at the pairwise distances, data point 0 was definitely closer to cluster B than any others, but for data point 1, clusters A and B were close calls. We'd like to say something like "data point 1 is most likely to belong to cluster A, but we don't want to completely rule out the possibility that it actually belongs to cluster B."

Pairwise distances	Centroid of Cluster A	Centroid of Cluster B	Centroid of Cluster C	Cluster Assignment
Data point 0	7.071	4.472	6.083	Cluster B
Data point 1	3.162	4.472	5.385	Cluster A
Data point 2	3.000	5.000	1.414	Cluster C

How do we express the uncertainty? This is where the notion of **cluster responsibilities** come in. The cluster responsibility is the fraction of a data point being represented in a certain cluster. Let's take the assignment matrix we saw earlier:

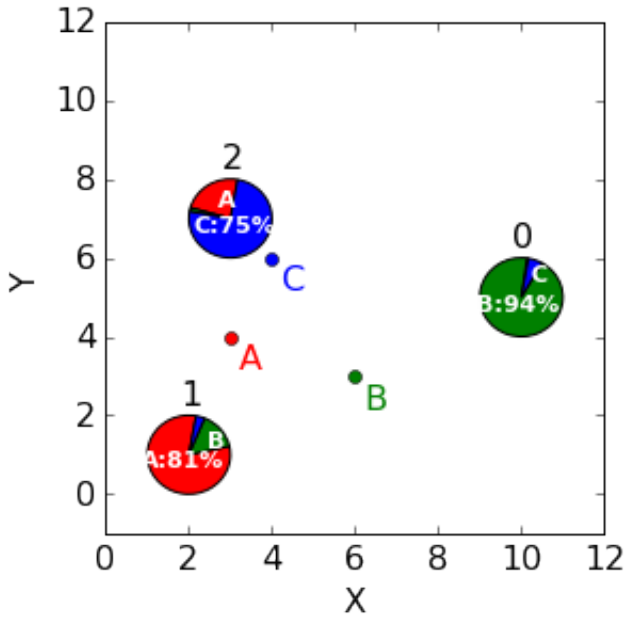
Assignment matrix	Cluster A	Cluster B	Cluster C
Data point 0	0	1	0
Data point 1	1	0	0
Data point 2	0	0	1
Member counts	1	1	1

... and replace 0's and 1's with fractions: (for now, trust us as to where these numbers come from; we'll get there soon)

Responsibility matrix	Cluster A	Cluster B	Cluster C
Data point 0	7	938	55
Data point 1	812	154	34
Data point 2	234	16	750
Soft counts	1.053	1.108	839

The first row tells us that data point 0 belongs to cluster B with 93.8% probability, an overwhelming certainty. (We shall use percentages instead of fractions here.) The next row expresses some uncertainty regarding the membership of data point 1, with 81.2% for cluster A and 15.4% for cluster B. The last row does so as well: 75.0% for cluster C and 23.4% for cluster A. These fractions are known as **cluster responsibilities**. For example, the responsibility of cluster A for data point 0 is 0.7%.

Unlike in k-means, where each point was assigned to one single cluster, we now have all points represented in all clusters, to a varying degree. For instance, 93.8% portion of data point 0 was represented in cluster A and the remaining portion in other clusters.



Finally, as in the assignment matrix, the column and row sums have meanings, too:

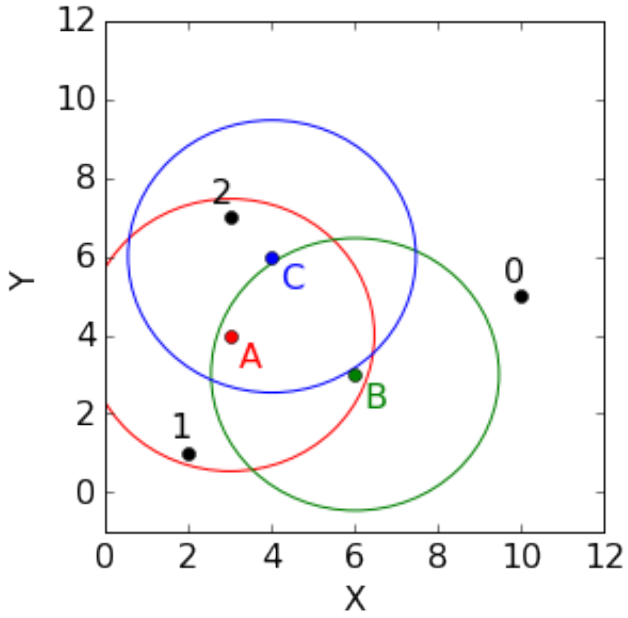
- Summing over a row always yields 1, as all fractions of a point must add up to a whole.
- Summing over a column yields what is known as the “soft count” of the cluster. This quantifies the total of the fractional representations of all points in that particular cluster. It’s a bit like counting the points within a cluster, except we’re talking decimals like 1.053.
- The soft counts for all clusters sums to the number of data points. In this example, 1.053+1.108+0.839=3.000.

E-step: Compute cluster responsibilities, given cluster parameters

How do we come up with the cluster responsibilities? **We look at how likely the data point is given the distribution of the cluster**. Recall from the lectures that each cluster in EM consists of a **cluster weight**, a **mean vector** (centroid) and a **covariance matrix**. The mean defines the center of the cluster, and the covariance defines the spread. The cluster weight represents the

relative representation of the cluster in the data. Summing all cluster weights yields 1. Finally, each cluster is modeled by a multivariate Gaussian distribution.

Let's re-visit the toy example above. We will keep the same set of means and add an "ellipse of uncertainty" around each mean, as follows:



Each ellipse visualizes the covariance matrix. In this case, all three clusters have the diagonal covariance [[3,0],[0,3]]. Since the off-diagonal entries are zero, the ellipses shown above are in fact circles. Also, since there is no reason to believe that one cluster would be better represented than other, let's assign a cluster weight of 1/3 to all clusters.

Note. The way we initialize covariances and weights may appear arbitrary. This is actually okay, just as initializing k-means with a random set of centroids is okay. We've just used an initial "estimate" of weights, means, and covariances. After computing the cluster responsibilities, we can update these parameters to arrive at a better estimate.

Let's take data point 0 and ask ourselves: How likely is the data point in the distribution of cluster A? The measure we'll be using is the **probability density function (PDF)** of the underlying Gaussian distribution. In SciPy, you may compute this measure using scipy.stats.multivariate_normal.pdf. At coordinates (10,5), i.e. data point 0, the PDF of the Gaussian distribution for cluster A has value 1.275e-5. To obtain this number, try running the following Python code block:

```
print multivariate_normal.pdf([10,5], mean=[3,4], cov=[[3,0],[0,3]])
>>> 1.275199678019219e-05
```

We need to scale this number by the cluster weight, as the cluster weight denotes the importance of the cluster compared to other clusters.

```
print 1/3.*multivariate_normal.pdf([10,5], mean=[3,4], cov=[[3,0],[0,3]])
>>> 4.2506655934e-06
```

The likelihood measure for cluster A at data point 0 is 4.251e-6. By itself, this number has no meaning -- we need to compute the same measure for clusters B and C and compare. Compute the PDF for cluster B and scale by the cluster weight:

```
print 1/3.*multivariate_normal.pdf([10,5], mean=[6,3], cov=[[3,0],[0,3]])
>>> 0.000630854709005
```

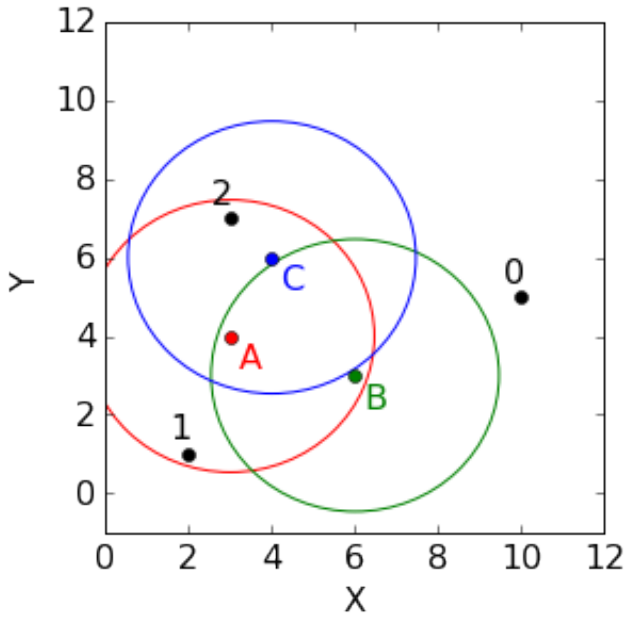
The likelihood measure for cluster B is 6.309e-4. Similarly, the likelihood measure for cluster C is 3.710e-5:

```
print 1/3.*multivariate_normal.pdf([10,5], mean=[4,6], cov=[[3,0],[0,3]])
>>> 3.71046481027e-05
```

Let's summarize them in a table:

	Cluster A	Cluster B	Cluster C
PDF of cluster at (10,5), multiplied by cluster weight	4,251E-03	6,309E-01	3,71E-02

Clearly, cluster B has the highest likelihood measure. This makes sense because data point 0 is closer to the center of cluster B than others.



The likelihood measure is all good, but it is typically a small number. Since we find it easier to reason with percentages than really tiny numbers, let's divide each value by the sum over all values, as follows:

Data point 0	Cluster A	Cluster B	Cluster C	Sum
Likelihood measure	4,251E-03	6,309E-01	3,71E-02	6,722E-01
Likelihood measure, divided by the sum	4.251e-6 / 6.722e-4 = **0.007**	6.309e-4 / 6.722e-4 = **0.938**	3.710e-5 / 6.722e-4 = **0.055**	

We're done: the numbers in the last row are the cluster responsibilities for data point 0! Notice that, by dividing through by the sum, we ensured that cluster responsibilities add up to 1. Let's review what we did so far:

- Compute the value of the PDF of the Gaussian distribution at each data point. Use the mean vector and covariance matrix as parameters of the Gaussian distribution.
- Multiply the PDF value by the cluster weight. This is the likelihood measure.
- Normalize all likelihood measures.

We invite you reproduce the steps for data points 1 and 2. You should get the following numbers:

Data point 1	Cluster A	Cluster B	Cluster C	Sum
Likelihood measure	3,34E+00	6,309E-01	1,408E-01	4,112E+00
Likelihood measure, divided by the sum	3.340e-3 / 4.112e-3 = **0.812**	6.309e-4 / 4.112e-3 = **0.154**	1.408e-4 / 4.112e-3 = **0.034**	

Data point 2	Cluster A	Cluster B	Cluster C	Sum
Likelihood measure	3,946E+00	2,742E-01	1,267E+01	1,689E+01
Likelihood measure, divided by the sum	3.946e-3 / 1.689e-2 = **0.234**	2.742e-4 / 1.689e-2 = **0.016**	1.267e-2 / 1.689e-2 = **0.750**	

We have just derived the responsibility matrix:

Responsibility matrix	Cluster A	Cluster B	Cluster C
Data point 0	7	938	55
Data point 1	812	154	34
Data point 2	234	16	750
Soft counts	1.053	1.108	839

M-step: Compute cluster parameters, given cluster responsibilities

Now that we've computed the cluster responsibilities, we must revise the parameters: cluster weights, means, and covariances. Even though the initial set of parameters was a wild guess, the updated parameters will be a better guess, as they will incorporate how the data points interact with the clusters. (Again, notice a parallel to k-means, where we start with guessing the centroids and fix them later using cluster assignments.)

Cluster weights. The relative importance of a cluster is determined by its soft count. Since the cluster weights must add up to 1, we add all soft counts and normalize them by their sum:

	Cluster A	Cluster B	Cluster C	Sum
Soft counts	1.053	1.108	839	3.000
Soft counts, divided by the sum	1.053 / 3.000 = **0.351**	1.108 / 3.000 = **0.369**	0.839 / 3.000 = **0.280**	

The soft counts, normalized by their sum, are the new estimates of cluster weights. In this example, the new estimates are 0.351, 0.369, and 0.280 for clusters A, B, and C, respectively.

Notice now that clusters A and B gained weight (from 0.333), while cluster C lost weight. According to the soft counts, cluster A and B are larger than the average cluster size, namely 1.

Means. We first add fractional parts of all data points, using the cluster responsibilities:

```
[Weighted sum of data points for cluster A]
= [Fraction of data point 0 represented in cluster A] * [data point 0]
  + [Fraction of data point 1 represented in cluster A] * [data point 1]
  + [Fraction of data point 2 represented in cluster A] * [data point 2]
= 0.007*[data point 0] + 0.812*[data point 1] + 0.234*[data point 2]
= 0.007*(10,5) + 0.812*(2,1)  + 0.234*(3,7)
= (0.063,0.035) + (1.624,0.812) + (0.702,1.638)
= (2.396,2.485)
```

Then we divide this sum by the soft count:

```
[mean of cluster A]
= (2.396,2.485)/1.053
= (2.275,2.360)
```

Similarly, for cluster B, we have

```
[Weighted sum of data points for cluster B]
= [Fraction of data point 0 represented in cluster B] * [data point 0]
  + [Fraction of data point 1 represented in cluster B] * [data point 1]
  + [Fraction of data point 2 represented in cluster B] * [data point 2]
= 0.938*[data point 0] + 0.154*[data point 1] + 0.016*[data point 2]
= 0.938*(10,5) + 0.154*(2,1) + 0.016*(3,7)
= (9.380,4.690) + (0.308,0.154) + (0.048,0.112)
= (9.736,4.956)
```

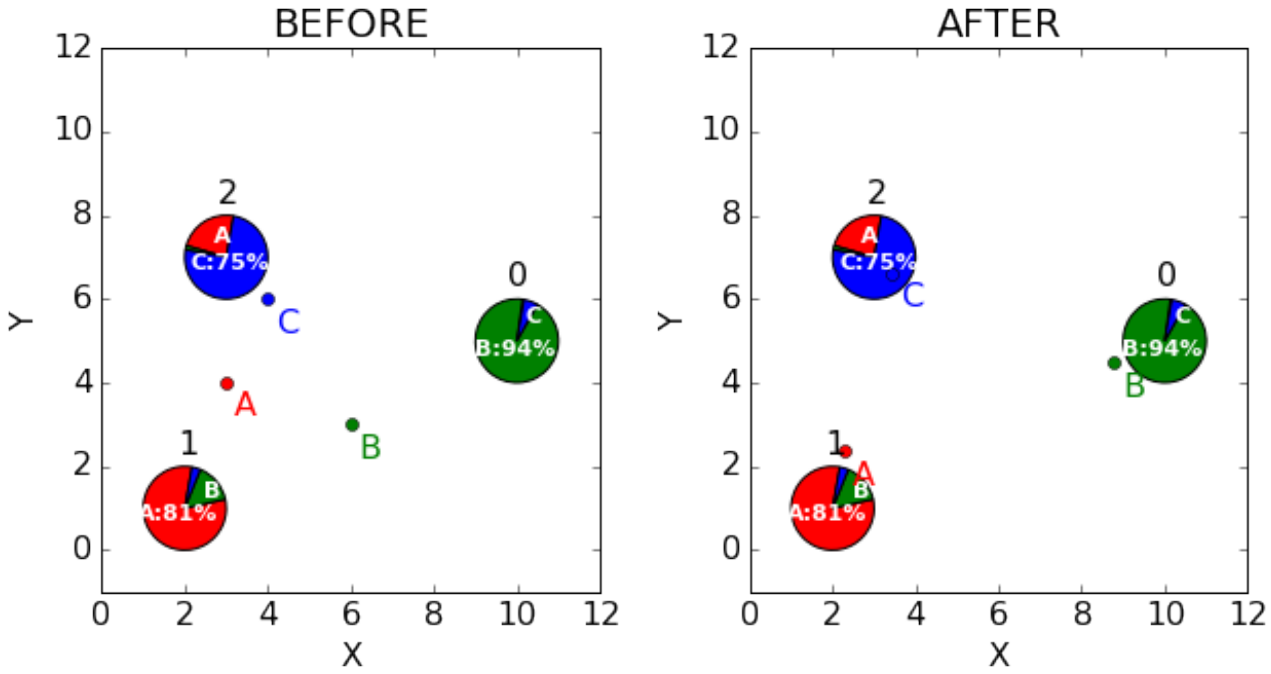
and

```
[mean of cluster B]
= (9.736,4.956)/1.108
= (8.787,4.473)
```

Repeating this process for cluster C, we obtain the new estimate of means:

New means	X	Y
Cluster A	2.275	2.360
Cluster B	8.787	4.473
Cluster C	3.418	6.626

Let's plot the old and new estimates of the means. Notice how the mean of cluster A moved closer to data point 1, for which cluster A is a majority. A similar scenario plays out for cluster B (moving towards data point 0) and cluster C (moving towards data point 2).



Covariances. The covariance is computed using fractional parts as well, except now we are summing matrices known as "outer products." For data point i and cluster k, we first compute the difference from the mean as follows:

$$x_i - \hat{\mu}_k$$

where x_i is the coordinates of the data point and $\hat{\mu}_k$ is the mean of cluster kThen we compute the "outer product" between the vector $x_i - \hat{\mu}_k$ and itself, as follows:

$$(x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T$$

Letting $x_i - \hat{\mu}_k$ to be a $d \times 1$ column vector, this product is a $d \times d$matrix. Taking the weighted average of all outer products gives us the covariance matrix, which is also $d \times d$.

Let's go through the calculations with outer products involving cluster A. First, we take the differences from the mean:

$x_0 - \hat{\mu}_A = (10,5) - (2.275,2.360) = (7.725,2.640)$
$x_1 - \hat{\mu}_A = (2,1) - (2.275,2.360) = (-0.275, -1.360)$
$x_2 - \hat{\mu}_A = (3,7) - (2.275,2.360) = (0.725,4.640)$

Then we compute three outer products. Note that the outer products are really two-by-two matrices.

$(x_0 - \hat{\mu}_A)(x_0 - \hat{\mu}_A)^T = [[7.725],[2.640]] * [[7.725,2.640]] = [[59.676,20.394],[20.394,6.970]]$
$(x_1 - \hat{\mu}_A)(x_1 - \hat{\mu}_A)^T = [[-0.275],[-1.360]] * [[-0.275, -1.360]] = [[0.076,0.374],[0.374,1.850]]$
$(x_2 - \hat{\mu}_A)(x_2 - \hat{\mu}_A)^T = [[0.725],[4.640]] * [[0.725,4.640]] = [[0.526,3.364],[3.364,21.530]]$

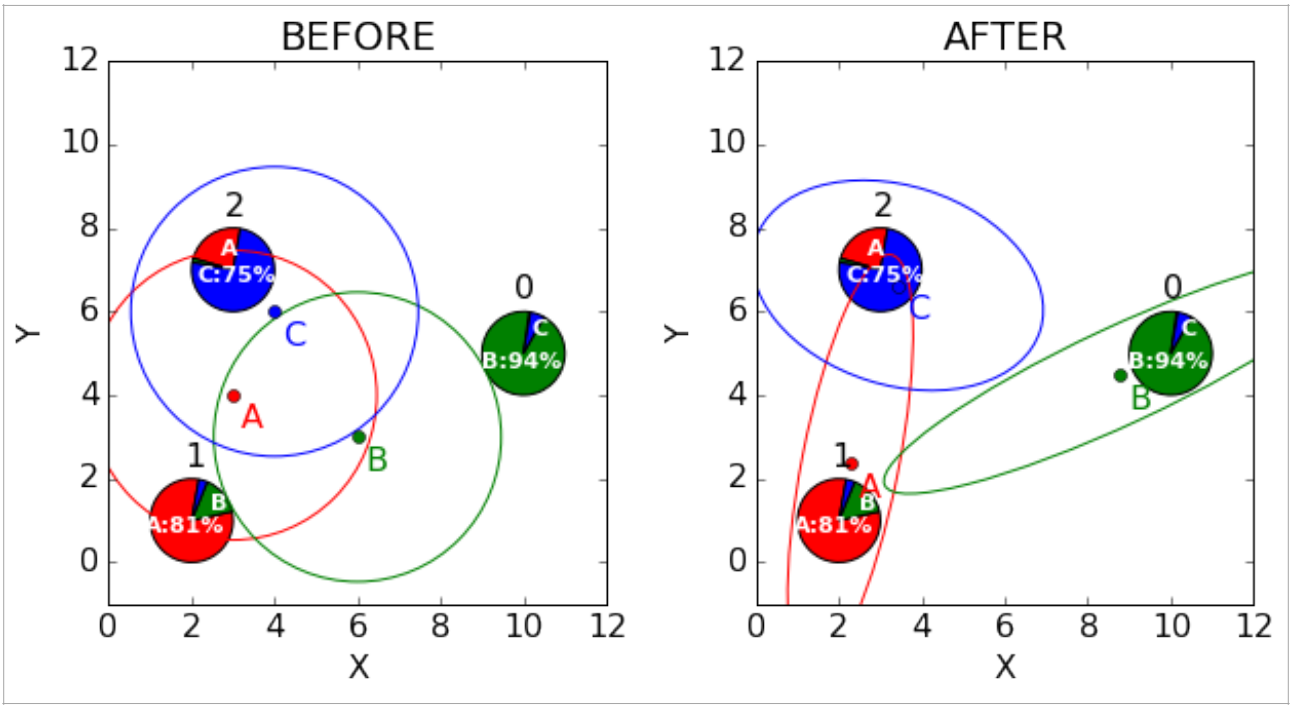
Finally, we take the weighted average, again using the cluster responsibilities:

```
[Weighted sum of outer products]
= [Fraction of data point 0 represented in cluster A] * [outer product for data
point 0]
  + [Fraction of data point 1 represented in cluster A] * [outer product for dat
a point 1]
  + [Fraction of data point 2 represented in cluster A] * [outer product for dat
a point 2]
=  0.007*[[59.676,20.394], [20.394,6.970]]
  + 0.812*[[0.076,0.374], [0.374,1.850]]
  + 0.234*[[0.526,3.364], [3.364,21.530]]
= [[0.602, 1.234], [1.234, 6.589]]

[New covariance for cluster A]
= [[0.602,1.234], [1.234,6.589]]/1.053
= [[0.572,1.172], [1.172,6.257]]
```

Repeating this calculation, we arrive at the following estimate of covariances:

New covariances	
Cluster A	[[0.572,1.172], [1.172,6.257]]
Cluster B	[[8.132,3.606], [3.606,2.004]]
Cluster C	[[3.078,-0.518], [-0.518,1.581]]



Phew! That was a lot of calculations. What was the effect of the update? First of all, all clusters have smaller regions of uncertainty -- overall, the covariance ellipses become significantly smaller in area. Secondly, the clusters changed shape to accommodate the data. For instance, cluster B becomes elongated in the direction towards data point 1. This is because a significant fraction of data point 1 (~15%) is represented in cluster B. For a similar reason, cluster A grows towards data point 2. **All in all, each time the means and covariances are revised, the clusters change shape to better reflect patterns in the data.**

EM algorithm: alternating between E-step and M-step.

Now that we have a better estimate of the cluster parameters, we again go back and recompute cluster responsibilities. Then we may come up with an even better parameter estimate. In fact, it turns out we can alternate between the E-step and M-step as many times as we want, and each time the quality of clustering improves.