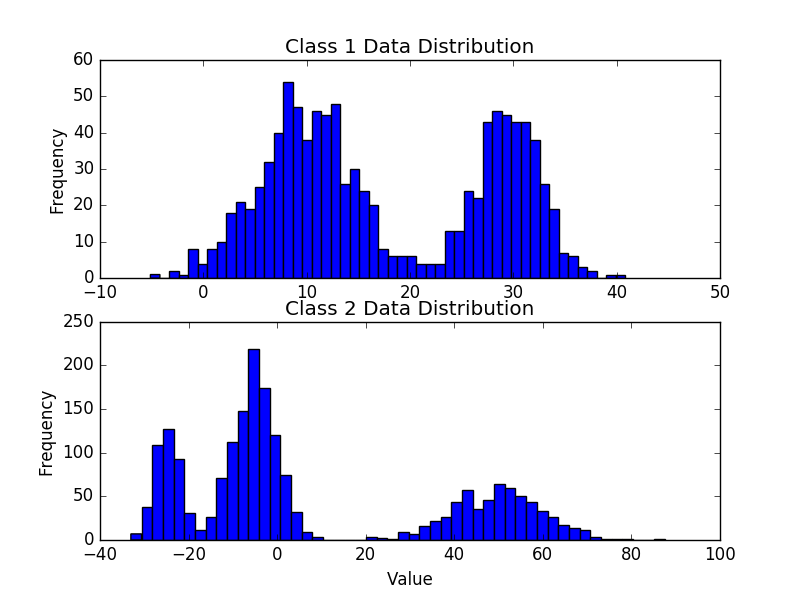
**Question 1**

Implemented in *gmm\_est.py*

**Question 2**



I plotted the above graph in order to discern the appropriate number of Gaussian components and parameter initializations. With this, I used the following values:

**Class 1**

mu\_init: [10, 30]

sigmasqinit: [7, 5]

wt\_init: [.6, .4]

**Class 2**

mu\_init: [-30, -10, 50]

sigmasqinit: [10, 13, 15]

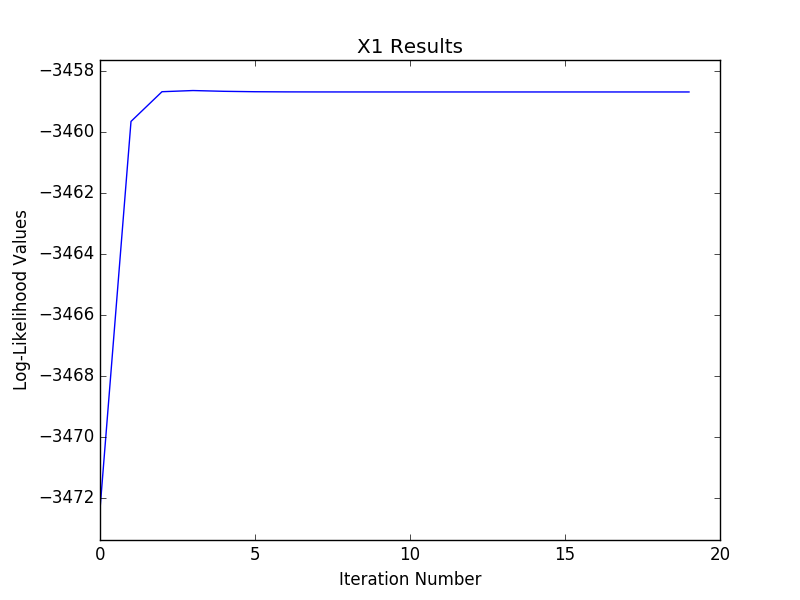
wt\_init: [.2, .5, .3]

**Final GMM Parameters for Class 1**

mu = [9.7748859236586476, 29.582587182965749]

sigma^2 = [21.922804563645315, 9.7837696128028284]

w = [0.59765463038822264, 0.40234536961263107]

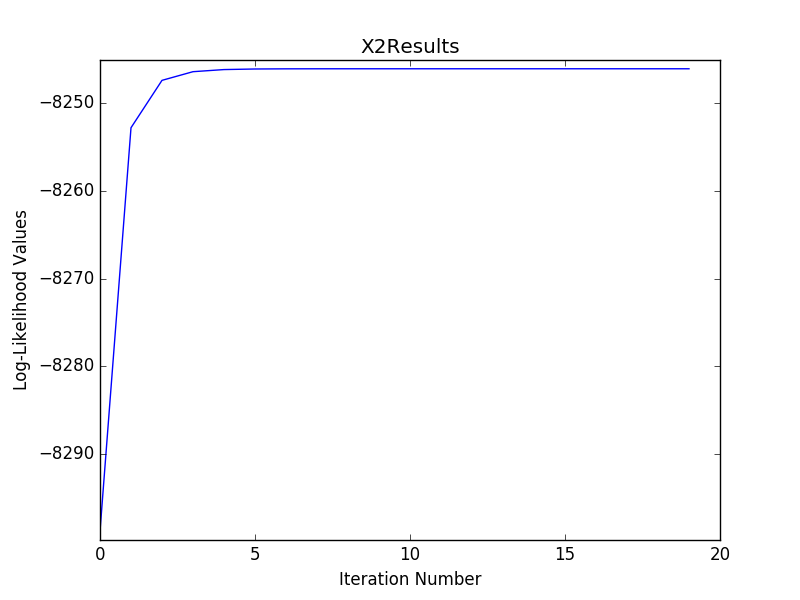


**Final GMM Parameters for Class 2**

mu = [-24.822751728709839, -5.0601582832398666, 49.62444471952756]

sigma^2 = [7.9473354076752969, 23.322661814417472, 100.02433750441168]

w = [0.20364945852681876, 0.49884302379613926, 0.29750751767685851]



I believe the program has converged because the curve flattens out and the values become relatively constant through the final iterations. Meaning, the exact values continue to get closer and closer until they reach a general constant point, as seen in the above graphs.

**Question 3**

Implemented in *gmm\_classify.py*. It can be run using the following command:

python gmm\_classify.py gmm\_test.py

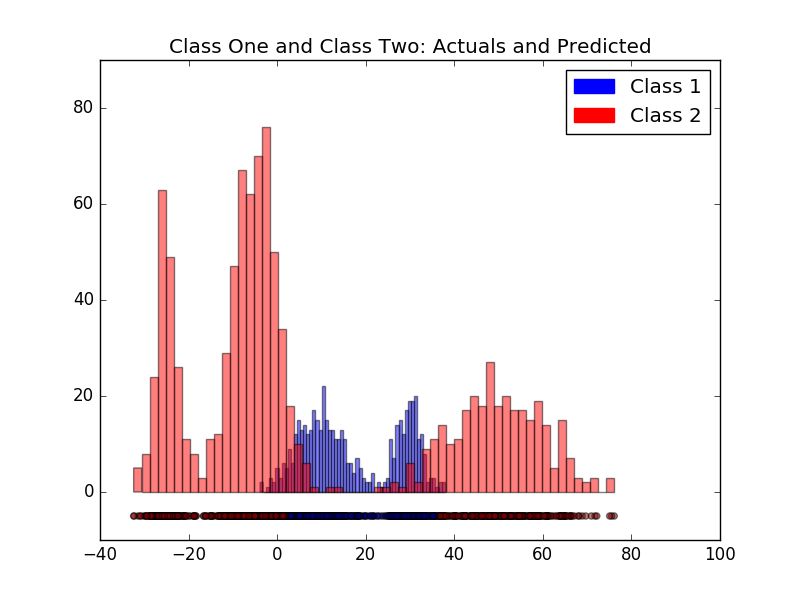
**Question 4:**

The GMM Model's Error: 0.0646666666667

The Prior Probability Model's Error: 0.444666666667

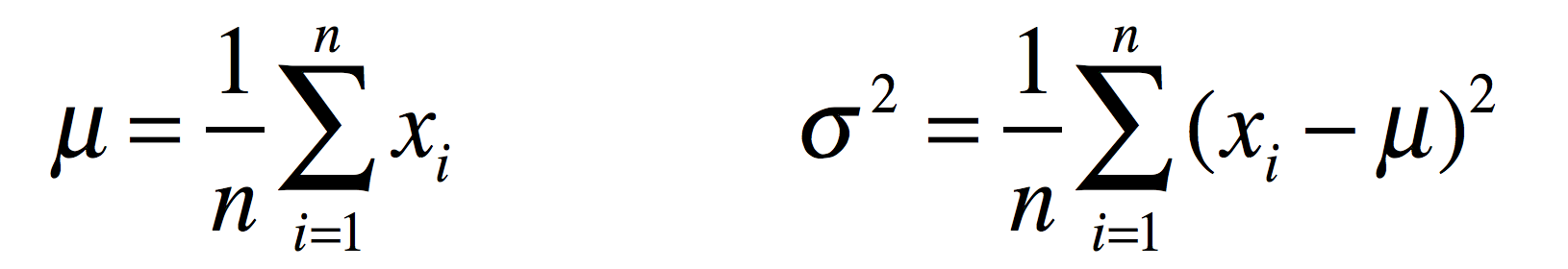
After running gmm\_classify on the test data, I determined that the GMM Model’s error was approximately 6.5%. I also implemented a prior probability model. Here, I found p1 (prior probability of class 1). Then, for each data point, randomly generated a float between 0.0 and 1.0. If the randomly generated number was less than p1 it was classified as 1, otherwise it was classified as 2. Given this implementation, when I ran the program the model had an error rate of approximately 44.5%. In turn, the GMM model has far better performance than the prior probability model.

Below is a graph, where the histograms are the ground-truth correct classes of the data, and the points plotted below are the data points as classified by my GMM classifier.



**Problem 5**

Yes, you would be able to use a closed-form solution to find the parameters of the GMM without using EM. Because each data point has been generated by exactly one Gaussian, a closed-form solution exists to maximize the probability of the data. In order to find the optimal parameters, you could take the partial derivatives with respect to the mean and standard deviation of the given Gaussian and set it to zero. Then, you can utilize the below equations for each Gaussian component. This works in this scenario because there are no instances in which a data point belongs to more than one Gaussian, and one can identify which Gaussian each data point belongs to. The Gaussians do not overlap such that data points are generated by more than one. In turn, the below closed-form solution using partial derivatives could compute the optimal solution.



**Problem 6**

Yes, in this scenario you would have to use Expectation Maximization because the closed-form solution only works in cases where you can map every data point back to its one and only Gaussian distribution. For instance, below illustrates a scenario where the Gaussian components overlap, and data points fall under both. Due to this ambiguity, the above closed form process cannot apply. Moreover, even in the case that the Gaussian components do not overlap, in order to apply the closed-form solution one must be able to associate the data points with their Gaussian component. In the presented scenario, both of these conditions are not met, and in turn EM must be used.