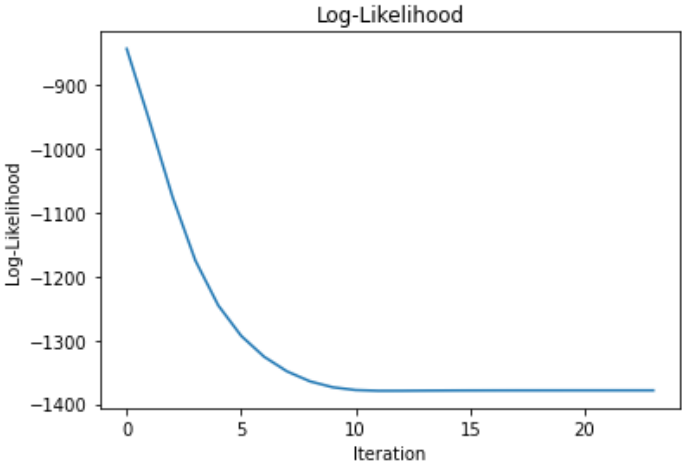
Implementing EM for MNIST dataset

The data was already only included the “6” and “2” digits and was scaled between 0 and 1, therefore the only step required was to use PCA to reduce the dimensional and project the original data into 4-dimensional vectors.

To implement the EM algorithm, first the parameters were initialized and the weights for the components and the posterior were initialized before starting the algorithm. A max number of iterations was set as a parameter in the *gmm* function, which acted as the iterable to start the EM algorithm. For each iteration, the posterior was updated for each component given the current parameters for each component (weight, mean, and covariance). Then, the weight, mean, and covariance for each component were updated given the posterior for each component. Afterwards, the expectation of log-likelihood was calculated, and if the log-likelihood converged (below a specified threshold) the loop stopped, otherwise, the loop continued until either the log-likelihood had converged or the maximum number of iterations had been reached.

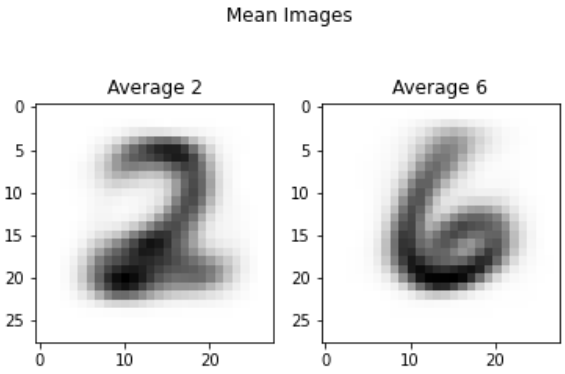
Below the plot of the log-likelihood at each iteration can been seen. Notably, it can be seen that the log-likelihood converged fairly quickly at iteration 23 (the maximum numbers of iterations was set at 100).



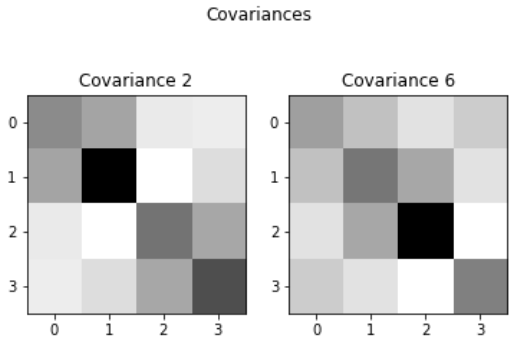
Below the weights for each component can be seen:

Weights
2: 0.4869125109088376, 6: 0.5130874890911624

Additionally, the mean of each component can be seen be reformatting the vector to view images of each mean. Notably, it can clearly be seen that the mean images consist of the “2” and “6” di —gits separately, showing a clear “average” image for each component.



Furthermore, the covariance matrix for each component can be seen below:



To infer the labels from GMM, the indices of the maximum posterior were found for each component, and were converted to the correct digit (2 or 6) based on the number of data points within each cluster. This was done by comparing the number of labels that matched in a certain range based on the known ranges of the 2 and 6 digits in the true labels. For example, if there were more "0" labels in the range 0:1032 (which corresponds to the "6" digits), then this meant that the "0" label could be inferred as the "6" digit. After doing this for both GMM and K-Means, the misclassification rates for each model with each digit can be seen below:

Misclassification Rates

	2	**6**
GMM	0.064922	0.009395
K-Means	0.061047	0.079332

Notably, GMM and K-Means had similar performance on the "2" digit with the misclassification rate of K-Means being slightly lower, but GMM performed much better than K-means on the "6" digit. Given this understanding, we can conclude that GMM achieved better performance than K-Means.