# EECE 5644: Assignment 3

Jeffrey Chen

November 15, 2024

Repository Link:
https://github.com/jeffreychen159/EECE-5644-Machine-Learning-and-Pattern-Recognition

---

# 1 Question 1

## 1.1 Generating Data

For my data distribution, I specified the parameters where the MAP classifier achieves around a 20% probability of error. Given that there requires $C = 4$ classes with uniform priors, I picked the following parameters:

$$priors = \begin{bmatrix} 0.25 & 0.25 & 0.25 & 0.25 \end{bmatrix}$$

$$meanVectors = \begin{bmatrix} 1 & 3 & 2 & 0 \\ 2 & 3 & 0 & 2 \\ 1 & 3 & 3 & 2 \end{bmatrix}$$

$$covMatrix_1 = \begin{bmatrix} 1 & 0.5 & 0 \\ 0.5 & 1 & 0.3 \\ 0 & 0.3 & 1 \end{bmatrix} \qquad covMatrix_2 = \begin{bmatrix} 1 & -0.3 & 0 \\ -0.3 & 1 & 0.4 \\ 0 & 0.4 & 1 \end{bmatrix}$$

$$covMatrix_3 = \begin{bmatrix} 1 & 0 & 0.2 \\ 0 & 1 & -0.2 \\ 0.2 & -0.2 & 1 \end{bmatrix} \qquad covMatrix_4 = \begin{bmatrix} 1 & 0.4 & -0.3 \\ 0.4 & 1 & 0.2 \\ -0.3 & 0.2 & 1 \end{bmatrix}$$

From these parameters, I generated 5 sets of training data with 100, 500, 1000, 5000, 10000 samples, and a test dataset of 100000 samples. These samples are listed in the repository.

## 1.2 Model Training

### 1.2.1 Data Format

To train the model, I needed to format the data generated. Given the data generated, there are 3 dimensions and 4 classes to be classified. I split the 3 dimensions as the input and the 4 class labels as the target. Since my target is only an integer, I categorized it using NumPy to allow the model to output 4 different class labels. After preparing the data for training, I developed the model.

### 1.2.2 Model Design

I decided to use Tensorflow and Keras to assist in developing my model. The model consists of two Dense layers, which represent MLP layers. Each of these layers has a relu activation and at the final layer, there is a softmax to ensure the values are positive and add up to 1. This was all compiled in sequential order.

### 1.2.3 Optimization and Data Gathering

To gather data on the model, the model complies built-in optimizers and loss calculations. For loss, I went with categorical_crossentropy and for optimizer, Adams was selected.

### 1.2.4 KFold Implementation

To implement the kFolds, I imported KFold from sklearn. This function allowed me to split the data into $k$ folds where when I run a for loop, it would loop $k$ times where each loop would select a set $\frac{1}{10}$ of the data that has not been previously chosen before, and be used as the validation data for that set. Each fold was trained 25 epochs deep and the final epoch was kept track of to analyze.

## 1.3 Predictions and Performance Analysis

### 1.3.1 Prediction Generation

For each fold, predictions were generated and once all 10 folds were generated and predicted, the average of probability for each classifier was generated and saved in the repository. By doing this, it allows each fold to be significant in the data processing and combines the 10 folds to give a value.

### 1.3.2 Results and Performance Analysis

The predicted values are saved in the repository with each representing the different amount of samples there were. On top of the predicted values, the loss and the accuracy were calculated by using the built-in functions of Keras and Tensorflow. Below is a table of each

sample size and the accuracy vs loss of both:

| samples | 100 | 500 | 1000 | 5000 | 10000 |
|---|---|---|---|---|---|
| accuracy | 74.0% | 76.4% | 79.1% | 80.0% | 80.1% |
| loss | 0.634 | 0.558 | 0.511 | 0.484 | 0.479 |

Figure 1: Table of the accuracy and loss of the training data.

From this table, we can see that as there are more samples being trained, it resulted in more accurate predictions. While it does pair more accurately, the accuracy does seem logarithmic where it seems to converge at about where the expected accuracy rate of 80% is. Below are the graphs of the performance results of training.

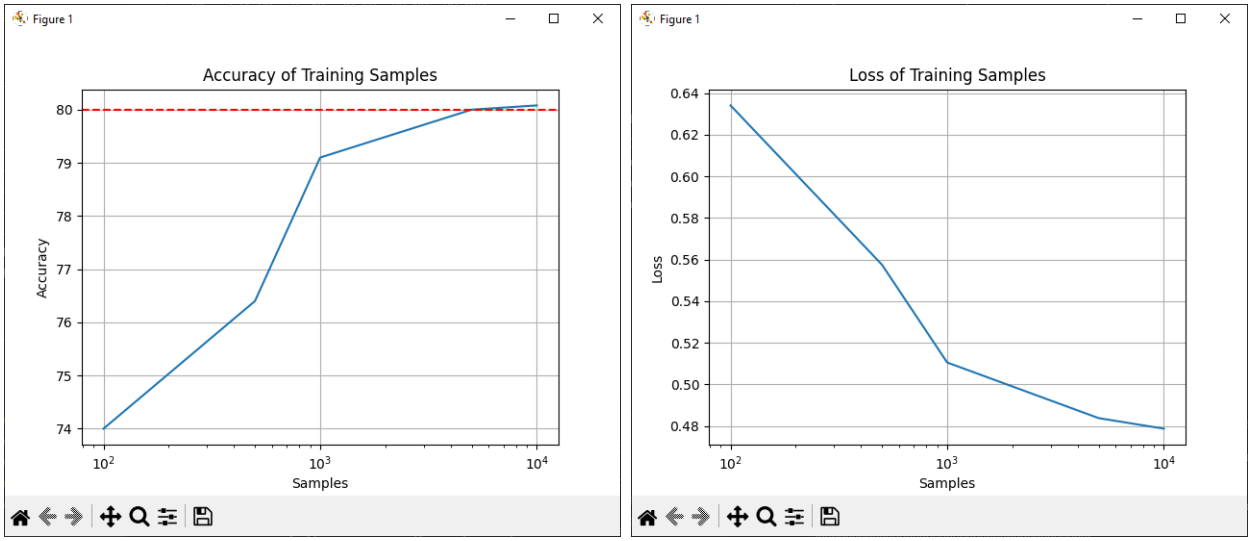


Figure 2: These graphs show the accuracy and the loss of the training determined by the amount of samples trained.

In the accuracy graph, the blue line indicates the accuracy over the samples. The red dash line indicates the rate where the accuracy starts to converge. In the loss graph, the blue line indicates the loss over samples.

From these graphs, we can see that as there are more training samples, there is less loss and greater accuracy in the predictions. This is to be expected as more training data should generate a higher accuracy.

On top of this data, when training each epoch, we can also see that training sets with more data start at a greater accuracy compared than the ones that have little data. But as each epoch trains, they do converge to a higher accuracy where generally, it seems to converge at somewhere between 74% to 80%.

# 2 Question 2

## 2.1 Data Generation

Below are the parameters used to generate my data.

$$priors = \begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 \end{bmatrix}$$

$$meanVector = \begin{bmatrix} 1 & 2 & 8 & 10 \\ 2 & 3 & 8 & 10 \end{bmatrix}$$

$$covMatrix_1 = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix} \qquad covMatrix_2 = \begin{bmatrix} 1 & -0.3 \\ -0.3 & 1 \end{bmatrix}$$

$$covMatrix_3 = \begin{bmatrix} 2 & 0.7 \\ 0.7 & 2 \end{bmatrix} \qquad covMatrix_4 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

## 2.2 Results and Comparison

### 2.2.1 Results

My code uses the prewritten code provided but instead, I just substituted the sample values to get my data. Through this code, I was able to get 3 sets of data for $N = 10, 100, 1000$, where $N$ is the number of samples. Below are the graphs of the Contours and the Log-Likelihood.

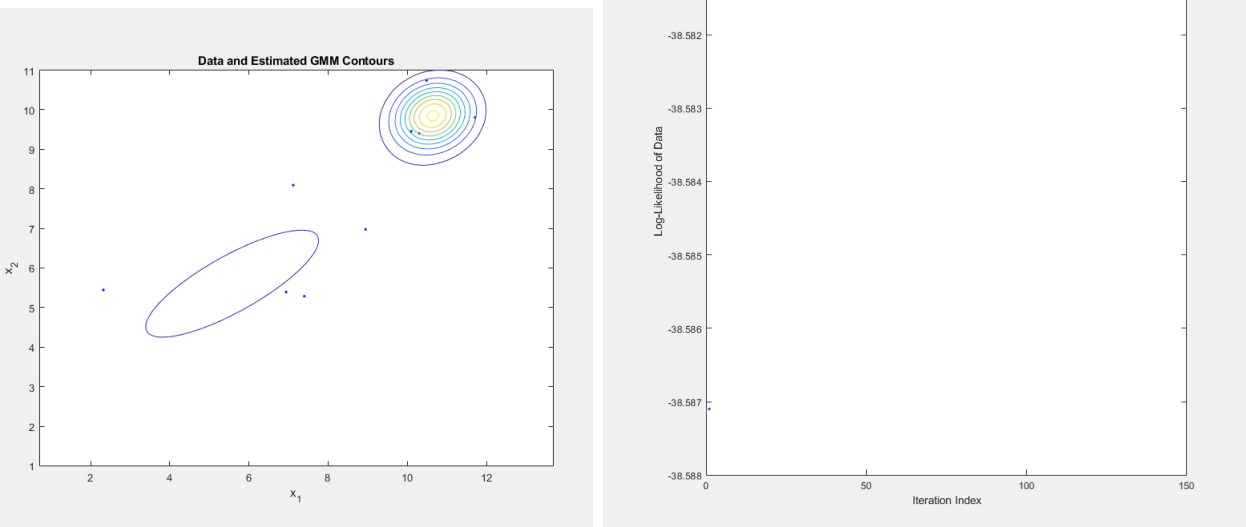


Figure 3: These graphs show the contours and the log-likelihood graph of $N = 10$

From what we can see in the contour graph, the one sitting at around $x_1 = 11$ and $x_2 = 10$ is where most of the clump is while there is another in the middle of the graph. Looking at the Log-Likelihood, we can see that this model doesn't seem to be too bad but a better sample size should give more details.

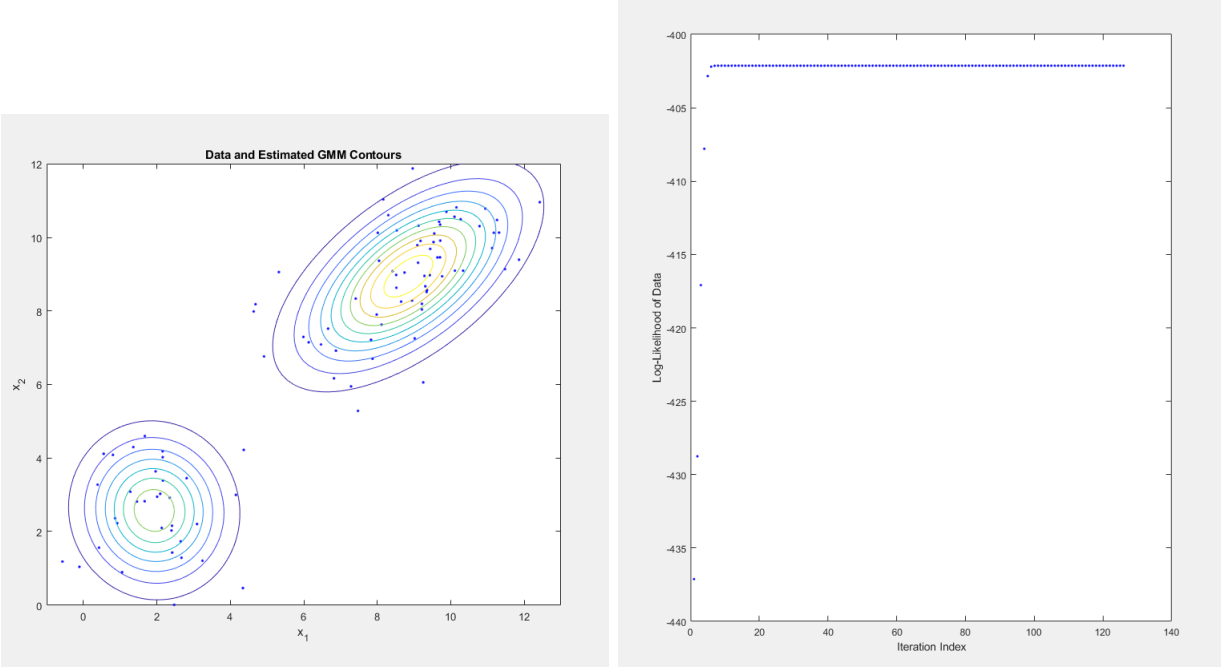


Figure 4: These graphs show the contours and the log-likelihood graph of $N = 100$

From what we can see in this contour graph, the contours seem to be more defined compared to the $N = 10$ graphs. This is probably due to more samples. On top of the contours, we can see that the Log-Likelihood takes more samples to converge which shows a more accurate representation of the model.

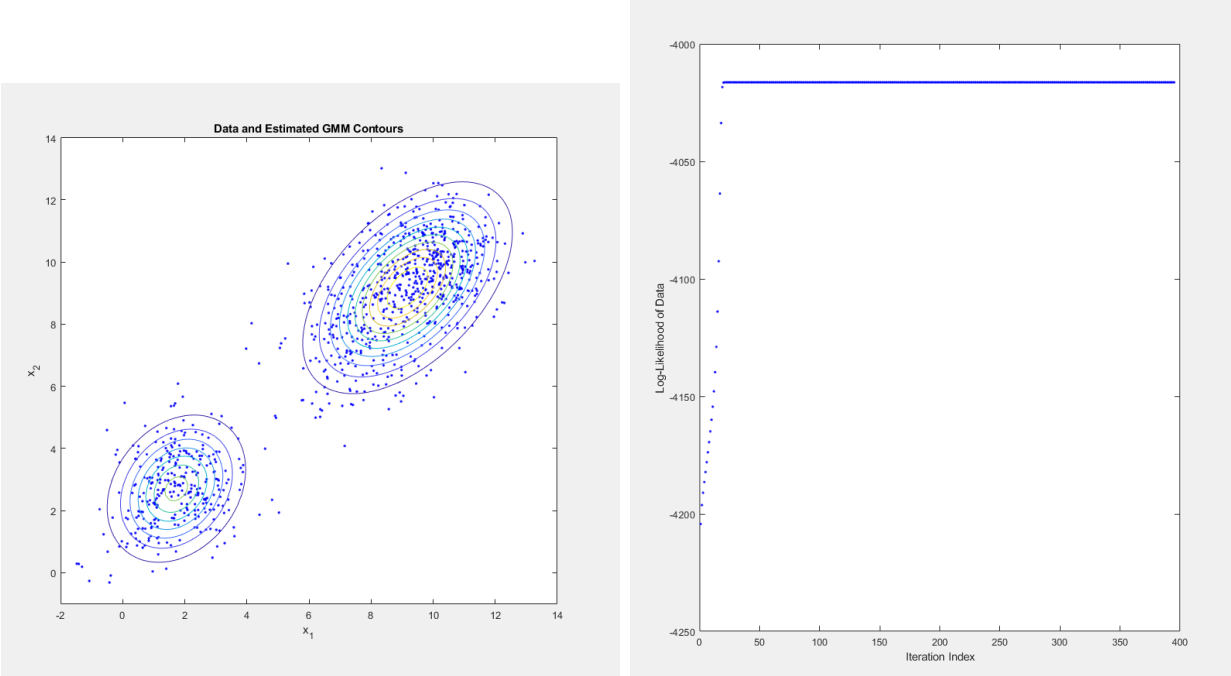


Figure 5: These graphs show the contours and the log-likelihood graph of $N = 1000$

From what we can see in this contour graph, the contours are definitely a lot more defined. We can see where the clumps are and how most are positioned in their respective circles. Looking at the Log-Likelihood data, it seems to take even more samples until it converges.

### 2.2.2 Comparison

Comparing the log-likelihood using a ratio test, we can compare how accurate each samples really is. We can do this by dividing the log ratio by the number of samples.

$$R_N = \frac{L}{N}$$

$$R_{10} = \frac{-38.6}{10} = -3.86$$

$$R_{100} = \frac{-402}{100} = -4.02$$

$$R_{1000} = \frac{-4020}{1000} = -4.02$$

Looking at these equations, technically the $N = 10$ was more accurate given the log-likehood ratio. But due to the low sample size, the log-likelihood ratio probably isn't a good representation of it's accuracy in this case due to a high error when it comes to having a low sample size.