Assignment 2 - Mahdiar Edraki

Thursday, October 22, 2020 8:07 AM

Question 1 (50%)

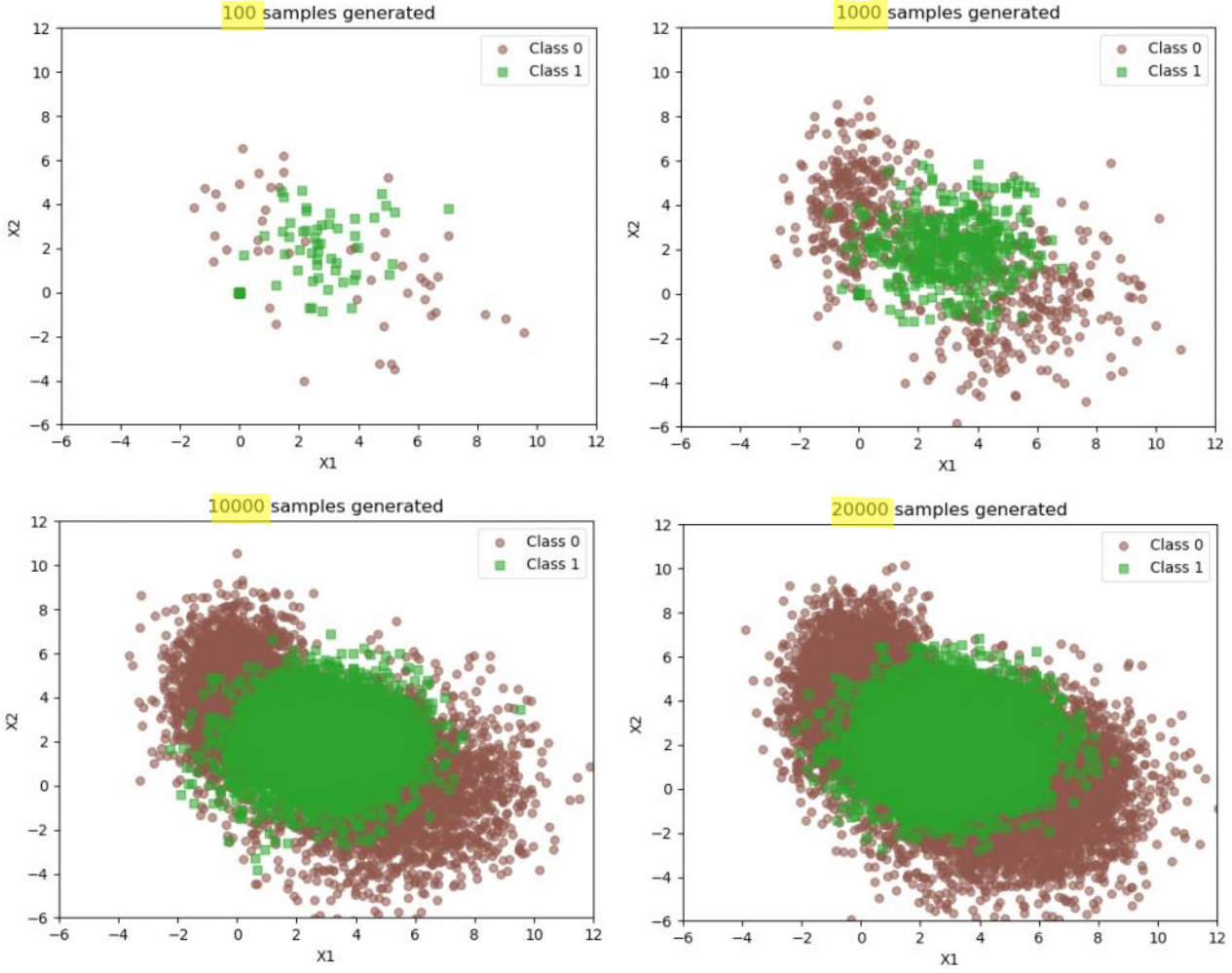
The probability density function (pdf) for a 2-dimensional real-valued random vector $\mathbf{X}$ is as follows: $p(\mathbf{x}) = p(\mathbf{x}|L=0)P(L=0) + p(\mathbf{x}|L=1)P(L=1)$. Here L is the true class label that indicates which class-label-conditioned pdf generates the data.

The class priors are $P(L=0) = 0.6$ and $P(L=1) = 0.4$. The class class-conditional pdfs are $p(\mathbf{x}|L=0) = w_1 g(\mathbf{x}|\mathbf{m}_{01}, \mathbf{C}_{01}) + w_2 g(\mathbf{x}|\mathbf{m}_{02}, \mathbf{C}_{02})$ and $p(\mathbf{x}|L=1) = g(\mathbf{x}|\mathbf{m}_1, \mathbf{C}_1)$, where $g(\mathbf{x}|\mathbf{m}, \mathbf{C})$ is a multivariate Gaussian probability density function with mean vector $\mathbf{m}$ and covariance matrix $\mathbf{C}$. The parameters of the class-conditional Gaussian pdfs are: $w_1 = w_2 = 1/2$, and

$$\mathbf{m}_{01} = \begin{bmatrix} 5 \\ 0 \end{bmatrix} \quad \mathbf{C}_{01} = \begin{bmatrix} 4 & 0 \\ 0 & 2 \end{bmatrix} \quad \mathbf{m}_{02} = \begin{bmatrix} 0 \\ 4 \end{bmatrix} \quad \mathbf{C}_{02} = \begin{bmatrix} 1 & 0 \\ 0 & 3 \end{bmatrix} \quad \mathbf{m}_1 = \begin{bmatrix} 3 \\ 2 \end{bmatrix} \quad \mathbf{C}_1 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

For numerical results requested below, generate the following independent datasets each consisting of iid samples from the specified data distribution, and in each dataset make sure to include the true class label for each sample. Save the data and use the same data set in all subsequent exercises.

- D_{train}^{100} consists of 100 samples and their labels for training;
- D_{train}^{1000} consists of 1000 samples and their labels for training;
- D_{train}^{10000} consists of 10000 samples and their labels for training;
- $D_{validate}^{20K}$ consists of 20000 samples and their labels for validation;



Part 1: (10%) Determine the theoretically optimal classifier that achieves minimum probability of error using the knowledge of the true pdf. Specify the classifier mathematically and implement it; then apply it to all samples in $D_{validate}^{20K}$. From the decision results and true labels for this validation set, estimate and plot the ROC curve of this min-P(error) classifier, and on the ROC curve indicate, with a special marker, indicate the point that corresponds to the min-P(error) classifier's operating point. Also report your estimate of the min-P(error) achievable, based on counts of decision-truth label pairs on $D_{validate}^{20K}$. Optional: As supplementary visualization, generate a plot of the decision boundary of this classification rule overlaid on the validation dataset. This establishes an aspirational performance level on this data for the following approximations.

The theoretically optimal classifier that achieves minimum probability of error is a 0-1 loss MAP classifier:

Since we have 2 classes:

Risk of deciding class label 0, given x

$$\begin{bmatrix} R(D=0|x) \\ R(D=1|x) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} P(L=0|x) \\ P(L=1|x) \end{bmatrix}$$

0-1 loss

Class posteriors

Once we find the Expected Risks, we pick the one with the lowest risk.

- First we need to find class posteriors:

Using Bayes Rule: $P(L=i|x) = \dfrac{p(x|L=i)\,p(L=i)}{p(x)}$

- $p(x) = \sum_{i=1}^{C} p(x|L=j)\,p(L=j)$ → equal scaling factor across all risks. Can be ignored for our purposes.
- $p(L=i)$ = class prior = given ✓
- $p(x|L=i)$ = gaussian pdf with μ_i and Σ ✓

Rewritten As:

$$\begin{bmatrix} R(D=0|x) \\ R(D=1|x) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} p(x|L=0)\,p(L=0) \\ p(x|L=1)\,p(L=1) \end{bmatrix}$$

Since we only have 2 classes, we can write the decision rule as a ratio of the expected risks of each class:

$$\frac{p(x|L_1)}{p(x|L_0)} \underset{D_0}{\overset{D_1}{\gtrless}} \left(\frac{p(L_0)}{p(L_1)}\right) \rightarrow \text{let } \gamma = \frac{p(L_0)}{p(L_1)}$$

$$\frac{p(x|L_1)}{p(x|L_0)} \underset{D_0}{\overset{D_1}{\gtrless}} \gamma$$

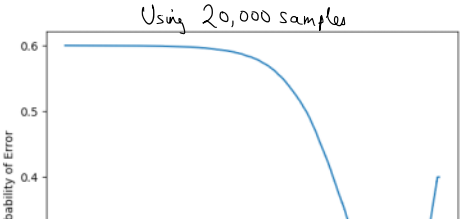
← Minimum expected risk classification rule

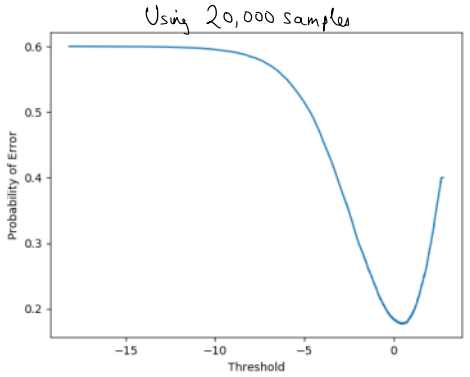
Therefore, the theoretically optimal classifier would have $\gamma = \dfrac{p(L_0)}{p(L_1)} = \dfrac{0.6}{0.4} = 1.5$

↳ we can also find the optimal threshold of maximizing the **log-likelihood** by:

$$\gamma = \ln(1.5) = 0.4055$$

Using 20,000 samples



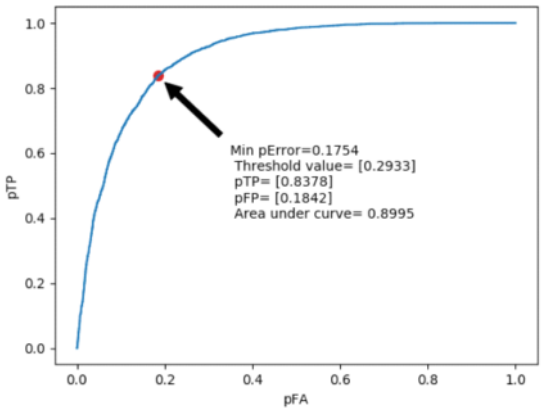


threshold of maximizing the log-likelihood by:

$\gamma = \ln(1.5) = 0.4055$

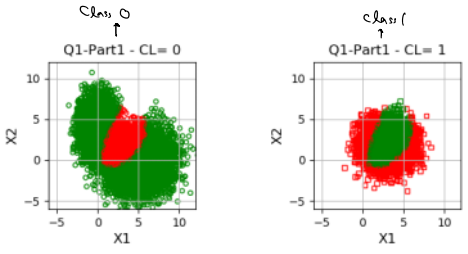
↳ This plot compares the probability of error as a function of the classifier threshold.
Our goal is to pick the threshold that minimizes the probability of error.
Another way to look at the plot above is to plot the **ROC curve**:

From the ROC curve, we can also identify the optimum threshold value.
The plot on the right indicates the point that corresponds to the min. P(error) achievable
Based on that results, the best classifier will have the following features:



- Minimum probability of error = 0.1754
- Threshold value, γ = 0.2933
- pTP = 0.8378
- pFP = 0.1842
- Area under ROC curve: 0.8995

Using the classifier above, we can classify the two classes as follows:



Part 2: (20%) (a) Using the maximum likelihood parameter estimation technique, estimate the class priors and class conditional pdfs using training data in D_{train}^{10000}. As class conditional pdf models, for $L = 0$ use a Gaussian Mixture model with 2 components, and for $L = 1$ use a single Gaussian pdf model. For each estimated parameter, specify the maximum-likelihood-estimation objective function that is maximized as well as the iterative numerical optimization procedure used, or if applicable, for analytically tractable parameter estimates, specify the estimator formula. Using these estimated class priors and pdfs, design and implement an approximation of the min-P(error) classifier, apply it on the validation dataset $D_{validate}^{20K}$. Report the ROC curve and minimum probability of error achieved on the validation dataset with this classifier that is trained with 10000 samples. (b) Repeat Part (2a) using D_{train}^{1000} as the training dataset. (c) Repeat Part (2a) using D_{train}^{100} as the training dataset. How does the performance of your approximate min-P(error) classifier change as the model parameters are estimated (trained) using fewer samples?

To estimate the class priors and class conditional pdfs, we need to estimate the weights and means and covariances of the Gaussian Mixture Models. This can be achieved by **maximizing the likelihood of the following parameters:**

Weight of each Gaussian pdf: $\alpha_l^{new} = \frac{1}{N} \sum_{i=1}^{N} p(l \mid x_i, \theta^g)$

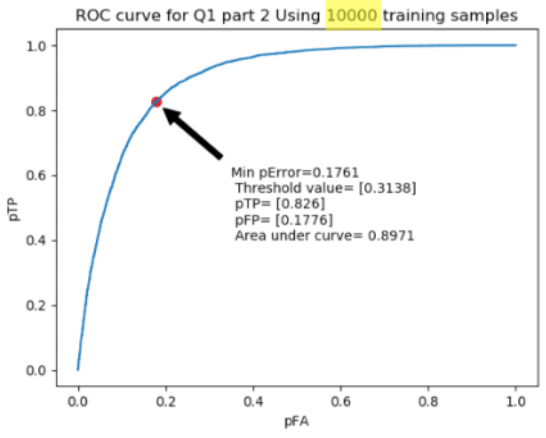
Mean of each Gaussian pdf: $\mu_l^{new} = \frac{\sum_{i=1}^{N} x_i \, p(l \mid x_i, \theta^g)}{\sum_{i=1}^{N} p(l \mid x_i, \theta^g)}$

Covariance of each Gaussian pdf: $\Sigma_l^{new} = \frac{\sum_{i=1}^{N} p(l \mid x_i, \theta^g)(x_i - \mu_l^{new})(x_i - \mu_l^{new})^T}{\sum_{i=1}^{N} p(l \mid x_i, \theta^g)}$

I use the python function `sklearn.mixture.GaussianMixture` and obtain the following estimates for the Gaussian mixture models:

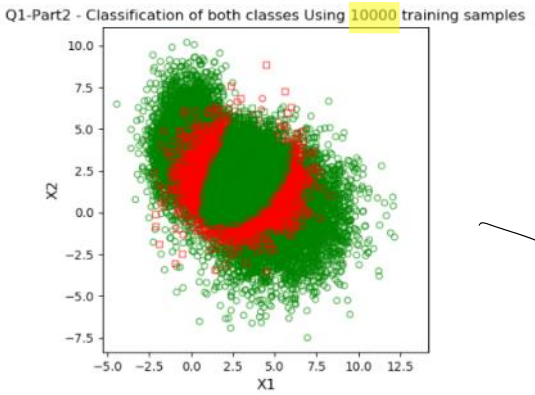
Q1 Part 2 a) Using D_{train}^{10000}

ROC curve:

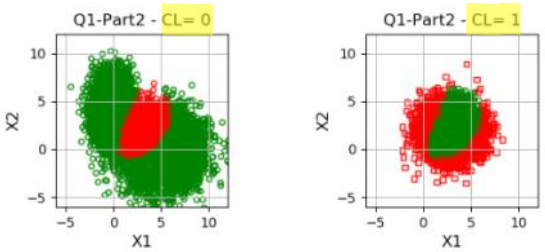


From the ROC curve we get **minimum probability of error = 0.1761**

Using threshold $\gamma = 0.3138$, we get the following class label classification of the $D_{validate}^{20k}$ dataset:

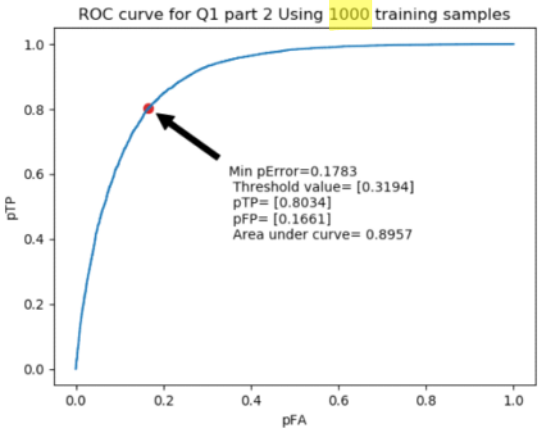


Separated into each class label



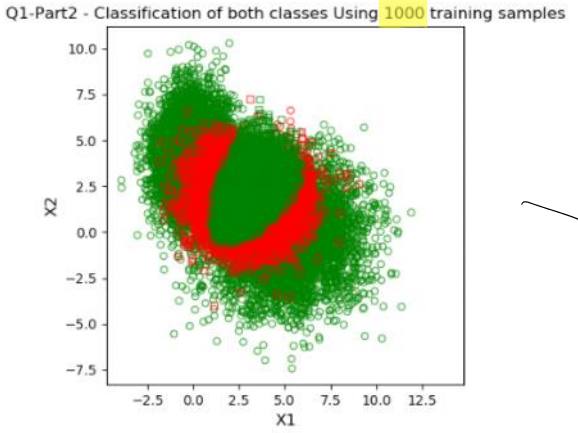
Q1 Part 2 b) Using D^{1000}_{train}

ROC curve:

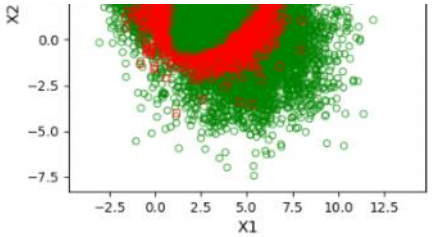


Using D^{1k}_{train}, we get **minimum probability of error = 0.1783**

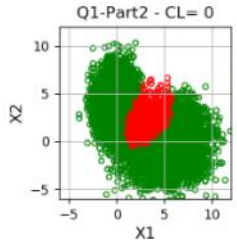
Using threshold $\gamma = 0.3194$, we get the following class label classification of the $D^{20k}_{validate}$ dataset:

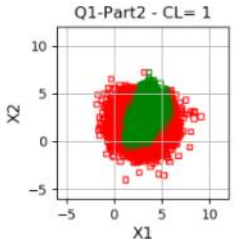


Separated into each class label



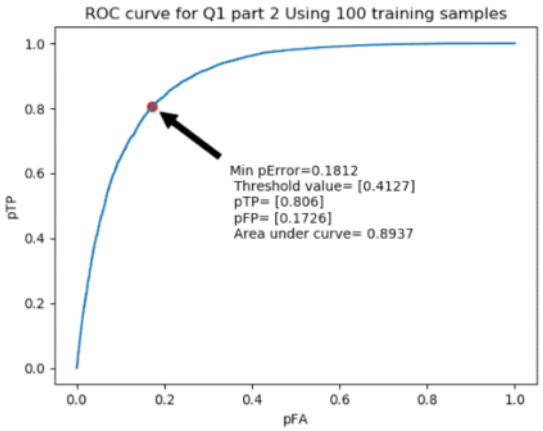
Separated into each class label





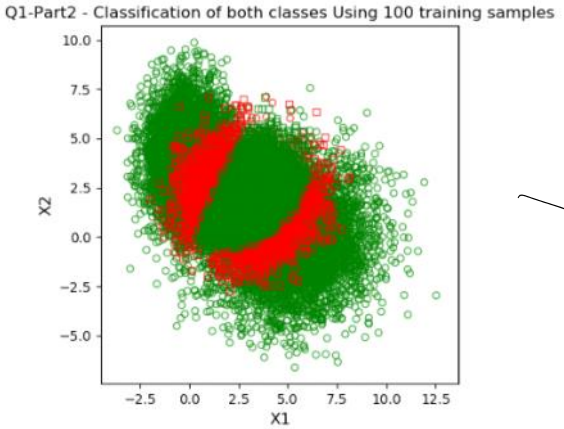
Q1) Part 2) c) Using D^{100}_{train}.

ROC curve:

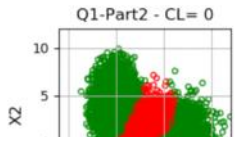


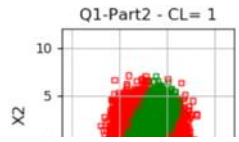
Using D^{100}_{train}, we get **minimum probability of error = 0.1812**

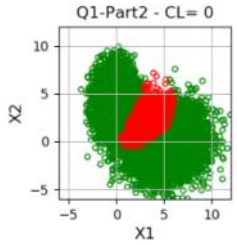
Using threshold $\gamma = 0.4127$, we get the following class labels classification of the $D^{200}_{validate}$ dataset:

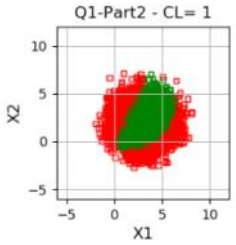


Separated into each class label









We can tabulate the results of minimum probability of error of the classifier obtained by the different **training dataset** and see which size dataset shows the lowest minimum p(error).

Training Dataset Size	20K	10K	1K	0.1K
Minimum Probability of Error	0.1754	0.1761	0.1783	0.1812
Threshold value	0.2933	0.3138	0.3194	0.4127

As expected, the **larger** the training dataset, the **smaller** the probability of error is. This is because, with more data, the algorithm can better estimate the true distribution of the classes.

Part 3: (20%) (a) Using the maximum likelihood parameter estimation technique train a logistic-linear-function-based approximation of class label posterior function given a sample. As in part 2, repeat the training process for each of the three training sets to see the effect of training set sample count; use the validation set for performance assessment in each case. When optimizing the parameters, specify the optimization problem as minimization of the negative-log-likelihood of the training dataset, and use your favorite numerical optimization approach, such as gradient descent or Matlab's *fminsearch* or Python's *minimize*. Use the trained class-label-posterior approximations to classify validation samples to approximate the minimum-P(error) classification rule; estimate the probability of error that these three classifiers attain using counts of decisions on the validation set. Optional: As supplementary visualization, generate plots of the decision boundaries of these trained classifiers superimposed on their respective training datasets and the validation dataset. (b) Repeat the process described in Part (3a) using a logistic-quadratic-function-based approximation of class label posterior functions given a sample.

Note 1: With $\mathbf{x}$ representing the input sample vector and $\mathbf{w}$ denoting the model parameter vector, logistic-linear-function refers to $h(\mathbf{x},\mathbf{w}) = 1/(1+e^{-\mathbf{w}^T\mathbf{z}(\mathbf{x})})$, where $\mathbf{z}(\mathbf{x}) = [1,\mathbf{x}^T]^T$; and logistic-quadratic-function refers to $h(\mathbf{x},\mathbf{w}) = 1/(1+e^{-\mathbf{w}^T\mathbf{z}(\mathbf{x})})$, where $\mathbf{z}(\mathbf{x}) = [1,x_1,x_2,x_1^2,x_1x_2,x_2^2]^T$.

Hint: The classifier designed in Part 1 uses the true pdf knowledge and achieves theoretically optimum performance in terms of minimizing P(error). The classifiers designed in Part 2 are approximations of the theoretically optimum classification rule using the correct functional form of the data pdf, however the quality of approximation for these generative models will get worse as their parameters are estimated with fewer training samples. The classifiers in Part 3 attempt to directly approximate class label posteriors, and the approximtion capability increases as the model gets more complex (linear to quadratic). The classifiers in Part 3, however, are limited by the approximation capability of their functional form. While the classifiers in Part 2 can asymptotically approach the performance level of the theoretically optimal one if they are trained with more data, the classifiers in Part 3 are bounded in performance by the fact that they can only generate linear or quadratic decision boundaries, so no amount of training data will enable them to asymptotically approximate the theoretically optimal classification rule (which has a classification boundary that is more complex than quadratic).

- Logistic-linear-function-based approximation of class label posteriors:

$$h_i(x,w) \approx p(l(x) = i \mid x)$$

$$h_i(x,w) = \frac{1}{(1+e^{-w^T z(x)})}$$ → where $z(x) = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$

and w is the model parameter

Given that we have 2 classes:

if $h(x,w) \approx p(L=1|x)$] we want to pick a w that **minimizes the negative log-likelihood of the class posteriors**
then $1-h(x,w) \approx p(L=0|x)$]

$$\rightarrow \hat{w}_{ML} = \underset{w}{\operatorname{argmin}} \; -\sum_{n=1}^{N} \ln\left(p(l_n|x_n,w)\right)$$

class label $\in \{0,1\}$

we can generalize $\ln(p(l_n|x_n,w)) = (l_n)\ln(h(x_n;w)) + (1-l_n)\ln(1-h(x_n;w))$

For a **Logistic-linear model**: with 2-Dimensional X:

$$w = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} \quad z(x) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

Objective: $$w_{ML}^{*} = \underset{w}{\operatorname{argmin}} \; -\frac{1}{N}\sum_{n=1}^{N}\left[(l_n)\ln(h(x_n;w)) + (1-l_n)\ln(1-h(x_n;w))\right]$$

Cost function →

where $$h(x_n;w) = \frac{1}{1+e^{-[w_0\, w_1\, w_2]\begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}}}$$

→ I will use python minimize to minimize the cost function.

- For the logistic-linear-function-based approximation of class labels, we are optimizing $\bar{w}$:

$$\mathbf{w} = [w_0 \;\; w_1 \;\; w_2]$$

↓ After optimizing on Python we get the following estimates for $\mathbf{w}$

```
Guess for linear fit parameters: w_0=-0.7489295803915598 w_1=0.10120473259234221 w_2=0.03496073233349854
```

$$\mathbf{w} \approx [-0.7489 \;,\; 0.1012 \;,\; 0.0349]$$

Recall that $h(x,w) \approx p(l(x)=1|x)$]

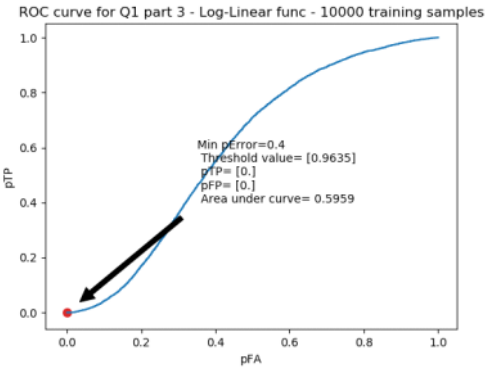
Recall that $h(x, w) \approx p(l(x) = 1 \mid x)$

and $1 - h(x, w) \approx p(l(x) = 0 \mid x)$

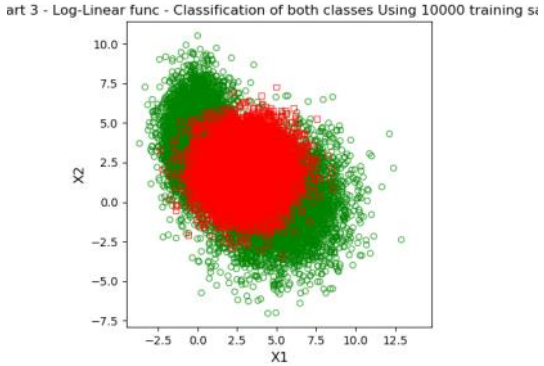
→ based on these functions, we can now implement the same algorithm as part 1 and 2 to classify the dataset:

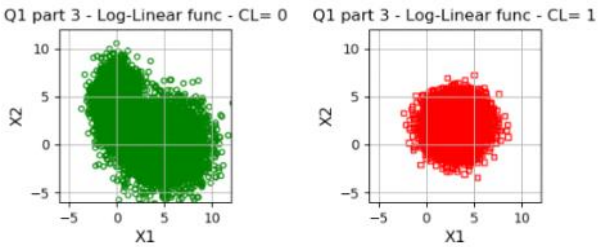
Q1) Part 3. a) Using D^{10k}_{train}:

ROC curve:



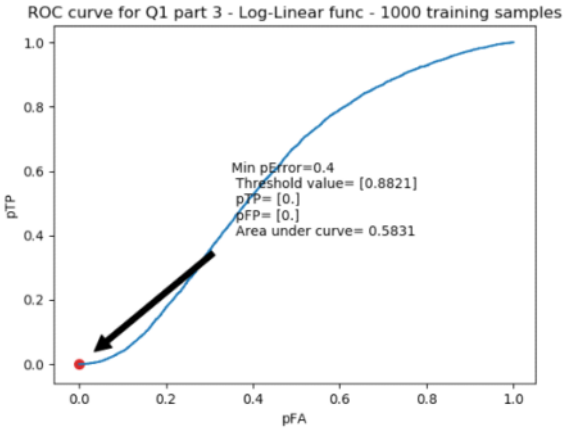
From the ROC curve we see that the logistic-linear-function is not a good classification model choice for our dataset. So the best result is to basically classify all data points as belonging to class label 0. We can run this classifier on our data to see its performance:



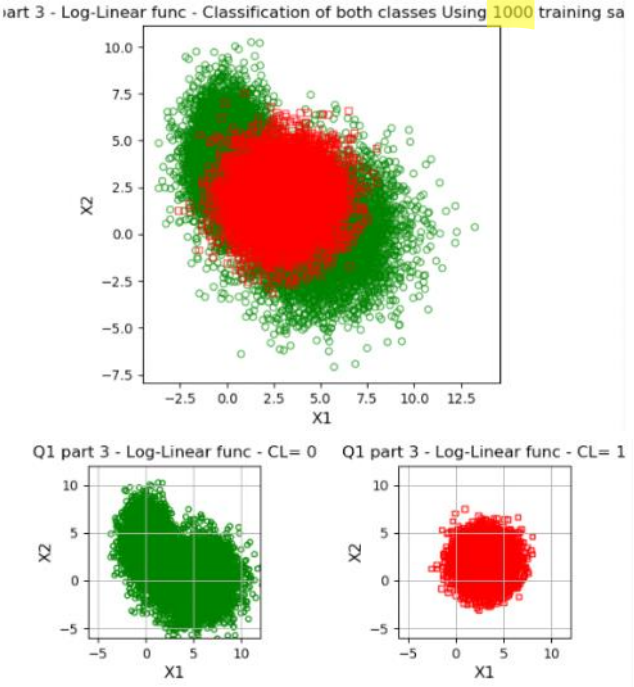


→ As expected, the classifier labels all data points as class label 0.

Q1) Part 3 - a) Using D^{1k}_{train} :

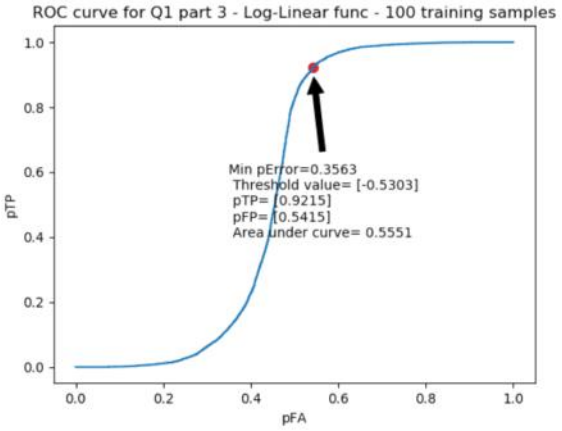


Similar to the previous dataset, we see that the model we have chosen to estimate class posteriors is not appropriate for our data distribution.

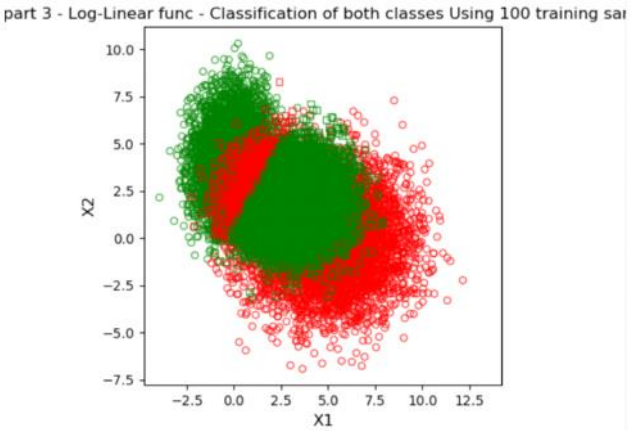


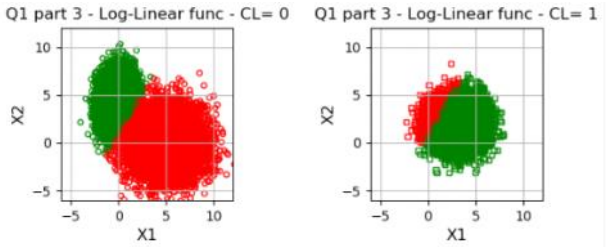
Again, all data points are labelled as class 0.

Q1) Part 3 a) D^{100}_{train} :



Interestingly, with a small dataset the logistic-linear-func optimizer is able to find a non-trivial line to classify the data as shown below:





↳ Here, we can finally see the **linear classification boundary** as it crosses through the data.

Based on the results above, we can tell that a logistic-linear function is not sufficient to model the class posterios of our class distributions.

In the next section below, I will use a logistic-quadratic function to estimate the class posterios and evaluate its performance.

Q1) Part 3 - b) Logistic-Quadratic-function

$h_i(x, w) \simeq p(l(x) = i \mid x)$

$$h_i(x,w) = \frac{1}{(1+e^{-w^T z(x)})} \rightarrow \text{where } z(x) = \begin{Bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_1 x_2 \\ x_2^2 \end{Bmatrix} \text{ and } w = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \end{bmatrix}$$

Given that we have 2 classes:

if $h(x,w) \approx p(L=1|x)$
then $1-h(x,w) \approx p(L=0|x)$

] we want to pick a w that minimizes the negative log-likelihood of the class posteriors

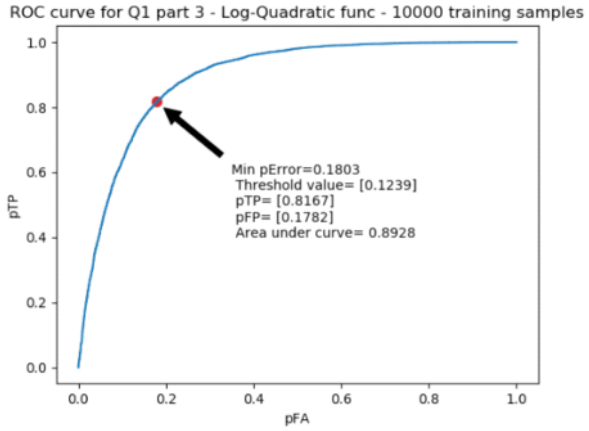
Objective: $$w_{ML}^{lok} = \underset{w}{\operatorname{argmin}} \; -\frac{1}{N} \sum_{n=1}^{N} \left[(l_n) \ln\left(h(x_n;w)\right) + (1-l_n) \ln\left(1-h(x_n;w)\right) \right]$$

Cost function

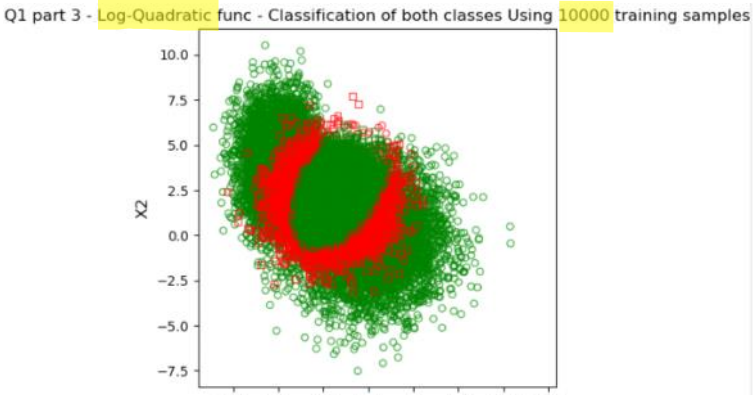
Similar to part a of this problem, I will use python minimize to find the optimum solution:

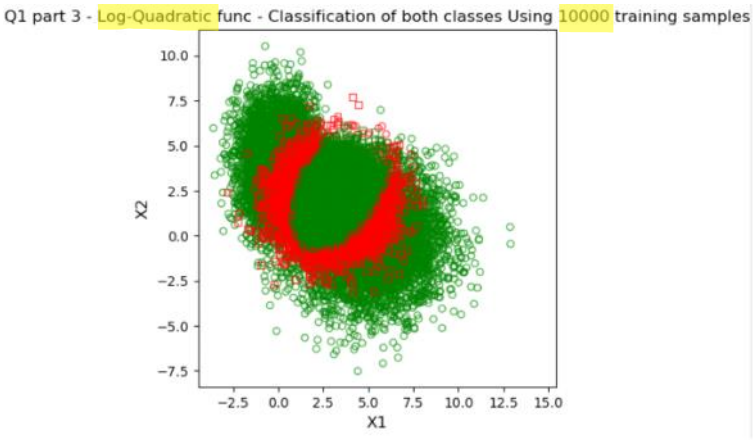
Q1. Part 3. b) Using D_{train}^{lok}:

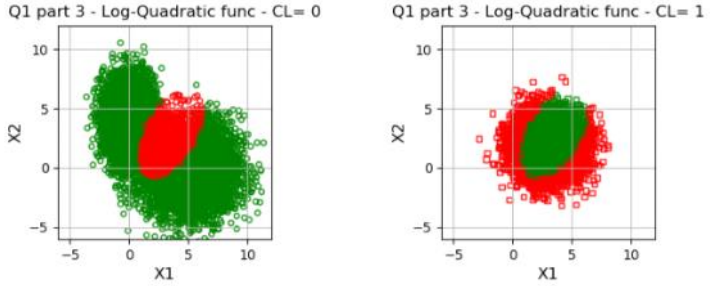
ROC curve:



Looking at the Roc curve, we can see that a logistic-quadratic function may have been a good choice for our dataset. Using the classifier developed above, we can implement and label our data as below:



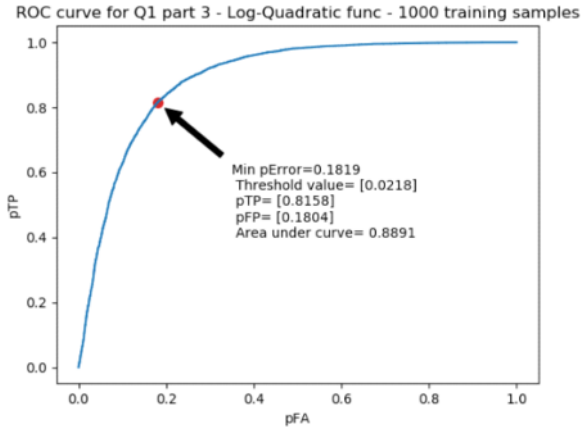


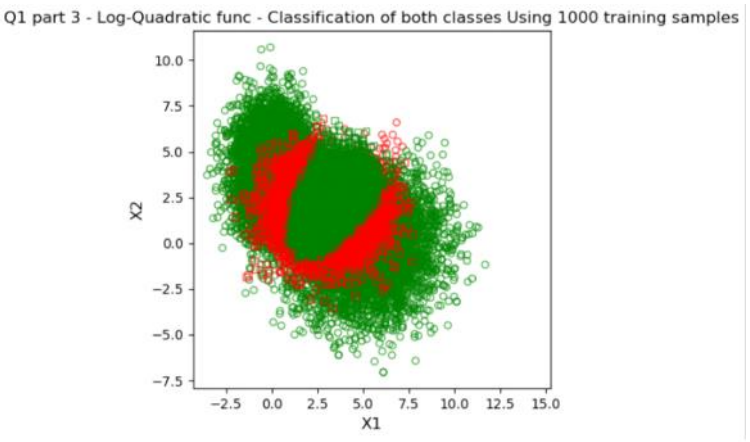


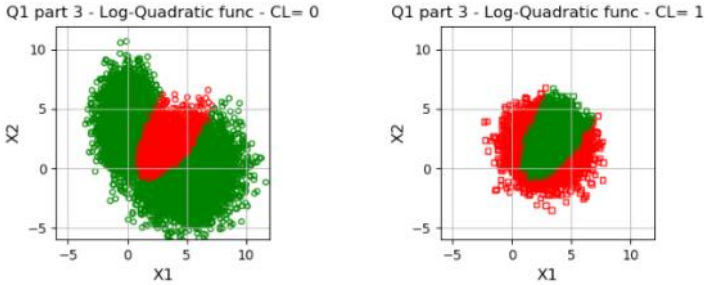
Here we can nicely see the classification boundary and it has a quadratic function shape. let us run the same function on smaller datasets and then compare their performance:

Q1_Part3_b) D^{1k}_{train} :

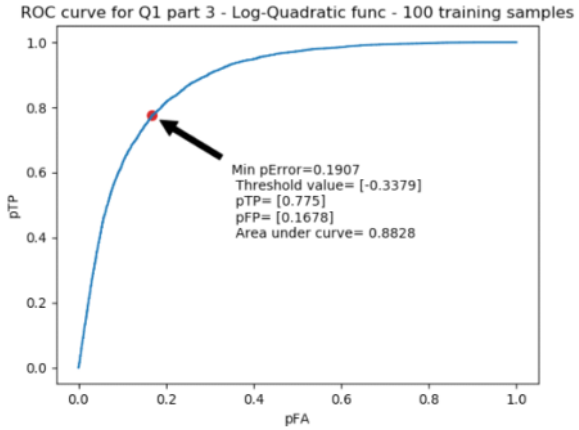
ROC curve:

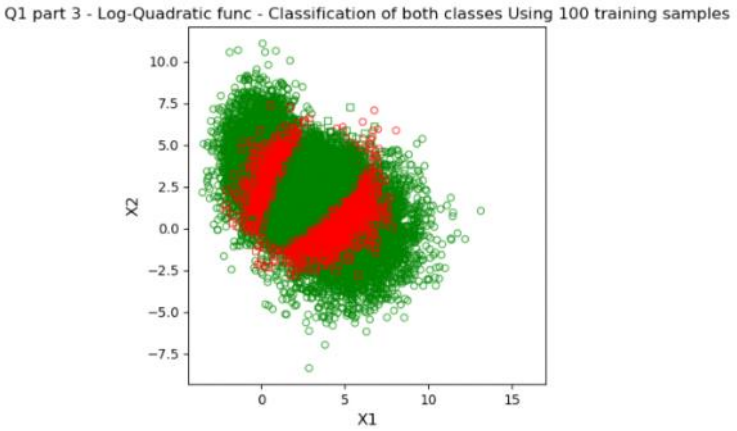


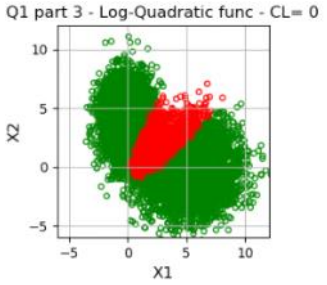


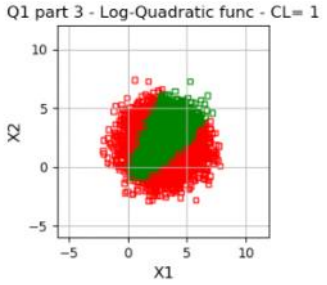


Q1 - Part 3 - b) D_{train}^{100} :









Now that we have obtained the minimum probability of error and area under ROC curve for all 3 datasets using logistic-linear and logistic-quadratic, we can add in those values to the table we developed in part 2, to compare the performance of all the classifiers:

Training Dataset Size	20K	10K	1K	0.1K
	Minimum probability of error			
Part 1 – Using **true data pdf**	0.1754			
Part 2 - Using **estimated data pdf**		0.1761	0.1783	0.1812
Part 3 – a) Logistic-**Linear** to estimate class posteriors		0.4	0.4	0.3563
Part 3 – b) Logistic-**Quadratic** to estimate class posteriors		0.1803	0.1819	0.1907

This table shows 2 things:

1) The more samples used to train a classifier, the better it will perform
2) Using a model to estimate the class posteriors that is most similar to the actual class posterior distribution yields the best results. (lower minimum probability of error)

Question 2 (50%)

The probability density function (pdf) for a 2-dimensional real-valued random vector **X** is a mixture of C Gaussian components with equal weights, distinct circularly symmetric covariance matrices, and mean vectors spaced evenly on a line. You may choose your own $C >= 10$, mean vectors, and covariance matrices for this problem.

Conduct the following model order selection exercise multiple (E) times (e.g., $E >= 100$), using both BIC and K-fold cross-validation (e.g., K = 10). Report a comprehensive (but compactly presented) summary of your experimental results regarding how BIC and K-fold cross-validation based model order selection decides on estimates of C (e.g., for both methods, show the histograms of estimated C values over multiple experiments).

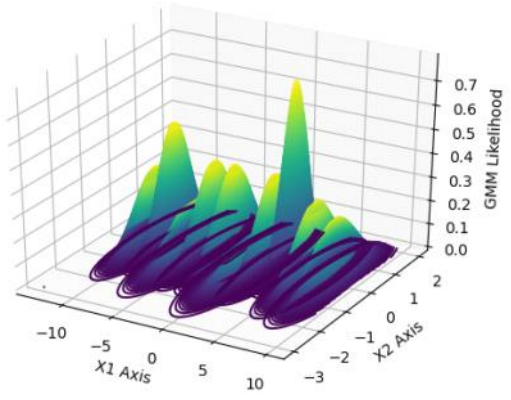
Repeat the following E times (i.e. conduct multiple experiments) using different values of M, number of components in your Gaussian Mixture Model (GMM) of the data pdf (e.g. M from 1 to 20 or higher if needed),...

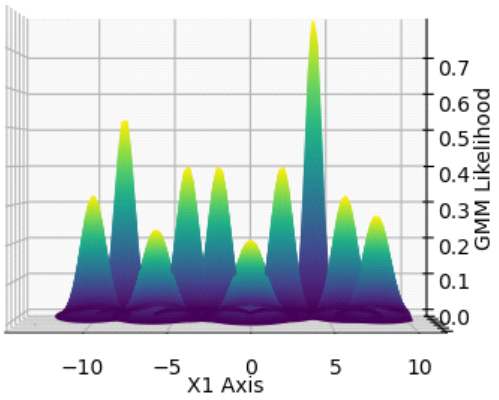
Generate multiple datasets from the true Gaussian mixture pdf you selected. Select your dataset sample counts as powers of 10 (e.g. from 10^2 to 10^6). Then, for each training set:

I am choosing C = 10, meaning my dataset is composed of 10 gaussian distributions with their means separated by +2 along x_1 direction. The covariances for each distribution was arbitrarily chosen in the vector Random_Sigma below:

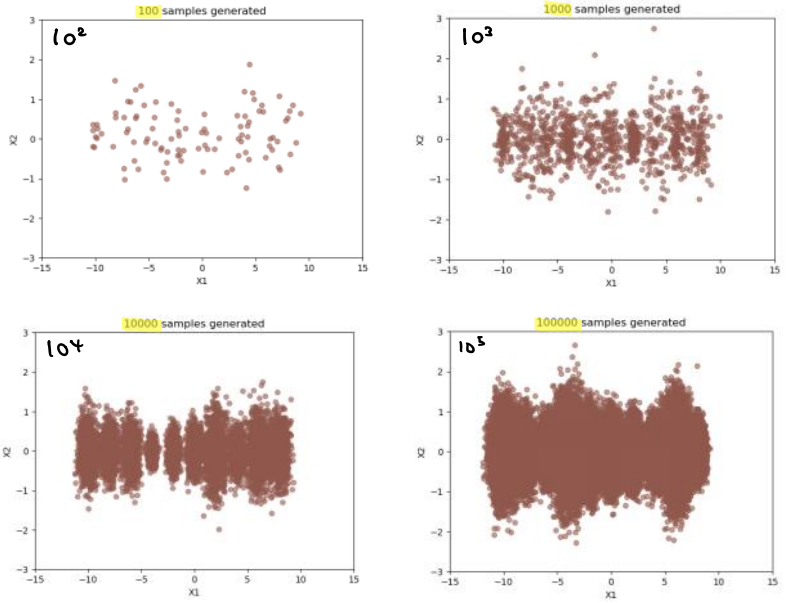
```
## GMM means and covariance ##
Sigma=np.zeros((C,n,n))
Mu=np.zeros((C,n))
Weight=np.zeros(C)
Random_Sigma=[0.5, 0.3, 0.7,0.4, 0.4, 0.8, 0.4,0.2,0.5,0.6]
Mean=-C
for i in range(0,C):
    Mean
    Sigma[i,:,:]=np.array([[Random_Sigma[i],              0],
                           [0,              Random_Sigma[i]]])
    Mu[i,:]=np.array([Mean,  0])
    Weight[i]=1/C
    Mean=Mean+2
```

Based on the defined GMM, we can obtain a plot of the likelihoods of each gaussian distribution and better see the amount of overlap of each distribution:





Now we can generate $N = 100, 1000, 10000, 100000$ datasets from the GMM above:



BIC (25%): For each M and for each dataset, use the EM algorithm to estimate the maximum likelihood parameter values for the M-component GMM, then evaluate the BIC for each M using the respective MLE-based GMMs. Select the best M (with smallest BIC value). Record the selected M values for each dataset in this experiment.

$$BIC = -2 \ln p_n(y, \hat{\theta}^*) + \ln(N) n$$

↳ in the equation above $N = N \times 2$ as we have 2 dimensions in our data set.

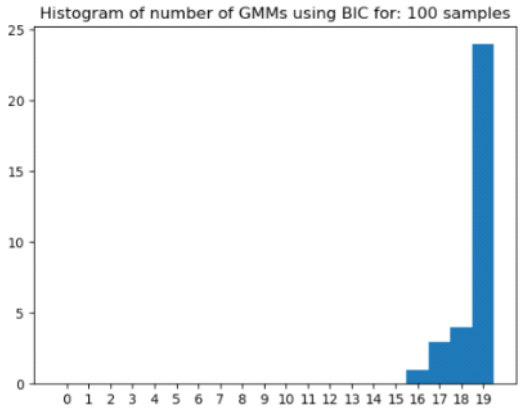
- and n = number of parameters in the M gaussian models used.

Each gaussian model for a 2D data has 6 parameters (2 for μ and 4 for Σ)

So $n = M \times 6$

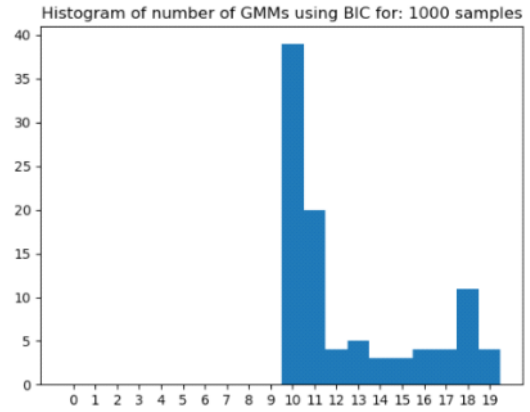
- I will vary M from 1 to 20 and see what is the best number of gaussian distributions that describe my data. Ideally that number should be 10

- Using D^{10^2} samples:



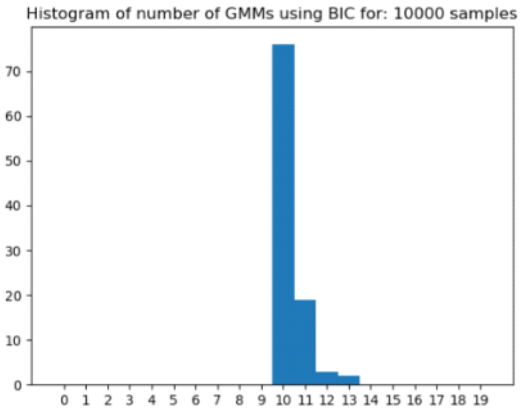
Using 100 samples, BIC estimates that we need 19 gaussian distributions to best describe the data. This is clearly much higher than the true number of distributions and I will discuss why that might be later down below.

Using D^{10^3} samples:



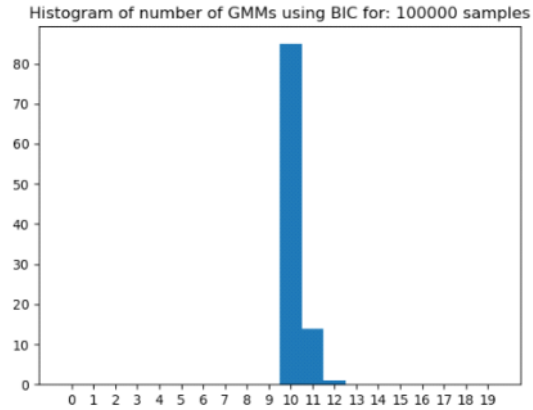
Using 1000 samples, BIC is already doing a pretty good job at correctly identifying the number of gaussian distributions of the data.

Using D^{10^4} samples:



With 10000 samples, BIC quite frequently correctly estimates the number of gaussian distribution

D^{10^5} samples:



With 100000 samples, BIC almost always correctly estimates the number of gaussian distributions that the data is comprised of.

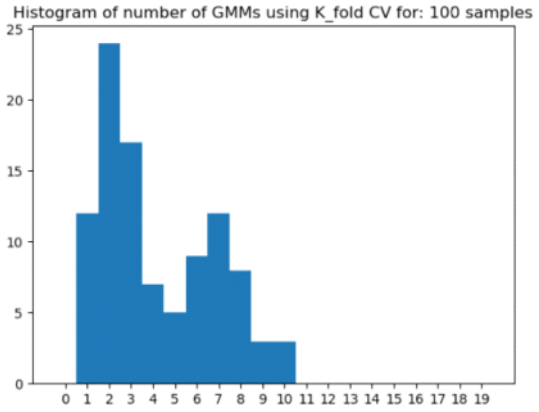
The results above show that BIC heavily trust the sample dataset it is given and assumes the true distribution is very similar to that of the sample dataset. Because of this, when a small sample set is fed into BIC, it tries to overfit the data and so it estimates much higher number of gaussian distributions describe the data than the reality.

K-fold cross-validation (25%): For each dataset, using your selected K, partition the dataset appropriately, then leaving each partition out once as validation set, do the following for each M: (a) use the EM algorithm to estimate the maximum likelihood parameter values for the for the M-component GMM with the training partitions, (b) then evaluate the log-likelihood of the data in the validation partition using the trained M-component GMM, (c) compute the weighted average of validation-log-likelihood values across K partitions (weighted proportional to sample counts in each validation partition, so you end up with a sample-average of log-likelihood where each sample contributes equally to the validation performance measure). Now that you have the average log-likelihood value for M-component GMMs with K-fold cross-validation procedure, select the best M as the one that maximizes this cross-validation log-likelihood value. Record the selected M values for each dataset in this experiment.

For K-fold cross validation, I will use K=5 and divide the dataset into K subdatasets. I will leave one data set out for the training stage and use that dataset to validate the trained classifier and obtain the M gaussians that maximizes

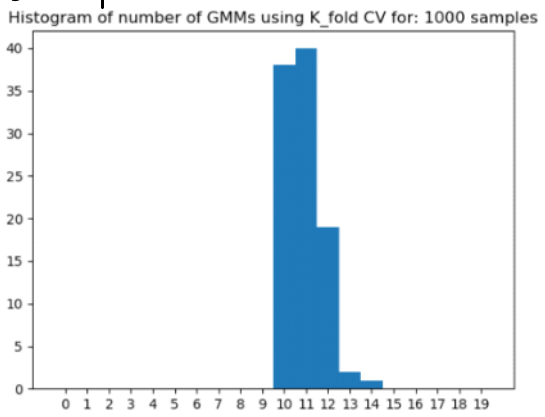
for the training stage and use that dataset to validate the trained classifier and obtain the M gaussians that maximizes cross validation. I will repeat this experiment 100 times and plot a histogram of the best estimated GMM number:

- For D^{10^2}:



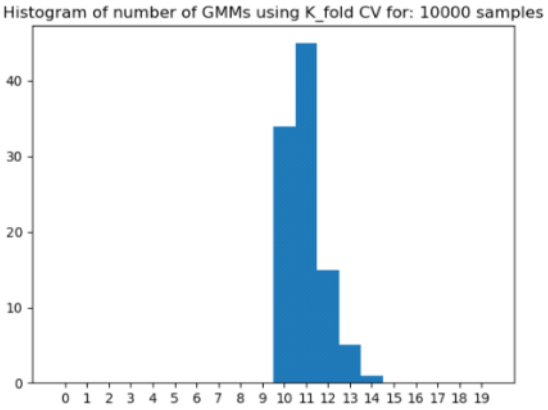
With 100 samples, K-fold CV estimates that we need 2 gaussian distributions to best describe the data. This is much lower than the true value.

Using D^{10^3} samples:



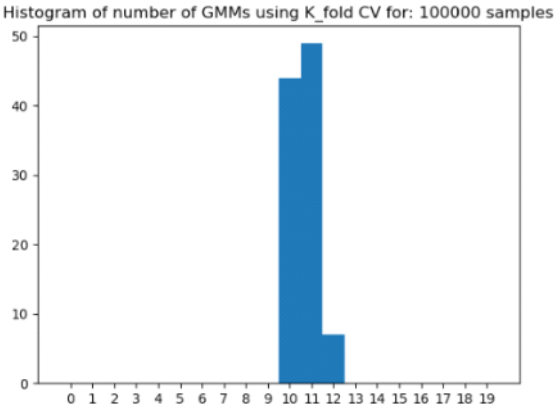
Using D^{10^3} samples, K-fold CV estimates we need 11 gaussian distributions to best describe the data which is a pretty good estimate

Using D^{10^4} samples:



With 10000 samples, Kfold CV estimates 11 gaussian distributions are needed again.

Using D^{10^5} samples:



With 100 000 samples, K-fold CV does a pretty good job at estimating the correct number of gaussian distributions needed to describe the data.

The results above show that K-fold CV tends to not trust the sample dataset as much as BIC does. Because of this, when we use smaller datasets, k-fold CV tries to underfit the data and so it estimates fewer than the correct number of gaussians are needed to describe the data.

Begin appendix on next page.