PRNN Assignment - 1

1. Question 1

Here we have to predict current health from given measurements as features.

1.1. Implementation

- At first datasets for P1 is loaded into colab and datas are separated into train and test sets
- Then training is done, as our loss fuction is mean squared loss so we have used the formulae:

$$W = (A^T A)^{-1} A^T y \tag{1}$$

1.2. Results

Here the output or testing is done on 3 things:

- Mean Squared error
- Mean absolute error
- P-Value out of significance test

The mean squared error on training data was 5.0596.
The mean squared error on test dataset was 5.0464
The mean absolute error on training data was 1.7917
The mean absolute error on test dataset was 0.8995
The p-value on train dataset is 1
The p-value on test dataset is 0.908

1.3. Observations

The output 'y' is linearly dependent on the given 2 features hence the error in this case is small as our assumptions matches to that of hidden density from which data is drawn. The above model was also tested in the iterative method with squared error loss and the error rapidly decreased in few epochs. Although in final code only the direct formulae based approach is shown. The p-value for our train and test data are pretty good which shows our hypothesis linear function is close to true function from which datas are picked.

2. Question 2

Here we have to predict life span of organism from given measurements of features.

2.1. Implementation

- At first datasets for P2 is loaded into colab and datas are separated into train and test datasets
- Here Y-labels are normalized to between 0 and 100 as the initial data values are in order of 10^6 but life span is generally of order 100, hence they are normalized.
- 3 different non linear models are created - polynomial,exponential and cosines.
- Training is done for each of the model and test and train errors for each of the given metrices are noted. Here also as only the features are non linear and it is linear in weights and hence the below mentioned formulae is used to calculate the optimum point : $W = (A^T A)^{-1} A^T y$

2.2. Results

Here the output or test data is done on 3 things Mean Squared error, Mean absolute error and P-Value out of significance test

Three non linear models are tried as follows:

- Model 1 (Polynomial model) : Here some extra polynomial features(like square of features, product of two features, product of three features) are added and trained.
- Model 2 (Exponential model): Here along with polynomial features, extra exponent of each features are added.
- Model 3 (Sine and cosine): Here along with polynomial features, extra sine and cosine of each features are added.

Train Output values

Model	MSE	MAE	P_value
1	30.05	4.14	1
2	19.21	3.12	1
5	25.9	3.74	1

Test Output values

Model	MSE	MAE	P_value
1	33.19	4.29	0.62
2	21.46	3.29	0.48
5	28.56	3.86	0.54

2.3. Observations

The p-values of hypothesis (on test data) is good for all models which implies our hypothesis of predicted data and actual data are nearer is true. The p-value of train data is 1 for each model which makes sense because our model is trained on those data. Further it is observed that the exponential model gave least error compared to all the models tried, because along with exponentials it also contained the quadratic term and probably because the function from which the above data is derived has exponential term in it. Non-linear model with sine and cosine term also did fairly well in comparision.

2.4. Problem faced

The error if 'y' is not normalised is very high(in the order of 10^{11}) which is understandable as y has very high value (in the order of 10^6) hence a small deviation in output will cause a large change in error value or squared error term. As 'y' represents lifespan of an organism hence it makes sense to normalize it and we have normalized it between 0 - 100, which reduced the magnitude of error significantly.

3. Question 3

Here we have data from 10 sensors and we have to use this data on various models to predict which product is being produced.

At first datasets for P3 is loaded into colab and datas are separated into train and test datasets Various models like bayes, logistic regressor are created and different metrices are computed.

3.1. Bayes Classifier with Normal density

3.1.1 Implementation

Here mean and variance both are assumed unknown and their ML estimate found for each class conditionals on train data and then tested on test data.

- The datas are separated into different classes and then their means and co-variance matrix calculated were calculated for each class.

- ML estimate for mean(μ_j) for each class (C_j)is

$$\mu_j = \frac{1}{n_j} \sum_{\forall i, y_i \in C_j} X_i \tag{2}$$

ML estimate for variance($\sum_j$) for each class (C_j)is

$$\sum_j = \frac{1}{n_j} \sum_{\forall i, y_i \in C_j} (X_i - \mu_j)(X_i - \mu_j)^T \tag{3}$$

Where n_j is number of datapoints of class j. X_i is the i^{th} data point(with 10 features) and μ_j is the mean vector of j^{th} class.

- For every data point class conditionals are calculated for each class using the below formulae.

$$g_j = \frac{1}{\sqrt{2\pi|\sum|}} e^{-\frac{1}{2}(x-\mu_j)^T \sum_j^{-1} (x-\mu_j)} \tag{4}$$

- The class where the class conditional is highest is choosen as the predicted class.

3.1.2 Results and Observations

We got an accuracy of 59.36 percent on test data. The F1 scores obtained are class 1: 0.54, class 2: 0.58, class 3: 0.64, class 4: 0.60, class 5: 0.59 . Fig 1 shows confusion matrix for this problem.

```
[[1627.  362.  264.  344.  374.]
 [ 380. 1742.  278.  268.  313.]
 [ 297.  298. 1914.  240.  291.]
 [ 321.  335.  255. 1794.  270.]
 [ 363.  310.  231.  302. 1827.]]
```

Figure 1

As expected the confusion matrix has large diagonal elemnts which imples more number of datas are being classified correctly.

3.2. Bayes Classifier with Laplacian Density

3.2.1 Implementation

We have taken naive bayes assumption here, so the parameters becomes median and variance term that are determined and then class conditionals are found for each class.

- Datasets are separated to different classes

- Here because of naive bayes assumption μ is median of each feature in each class which is found out by taking median of data points.

- By ML estimate each σ is given by:

$$\sigma_j = \frac{1}{n_j} \sum_{\forall i, y_i \in C_j} |x_i - \mu_j|$$

Where n_j is number of datapoints of class j.

- For every data point class conditionals are calculated for each class using the below formulae:

$$g_j = \frac{1}{\prod \sigma_j^i} e^{\frac{1}{2*\sigma_j}||x-\mu_j||_1}$$

As each features are independent hence for each class the densities corresponding to each features get multiplied and hence we get a product of σ terms in denominator.

3.2.2 Results and Observations

We got an accuracy of 48.48 percent on test data. The F1 scores obtained are class 1: 0.41, class 2: 0.46, class 3: 0.53, class 4: 0.49, class 5: 0.49 . Fig 2 shows confusion matrix for this problem.

```
[[1058.  505.  464.  429.  515.]
 [ 303. 1432.  428.  384.  434.]
 [ 250.  388. 1746.  287.  369.]
 [ 255.  438.  414. 1462.  406.]
 [ 270.  409.  409.  370. 1575.]]
```

Figure 2

3.3. GMMs

3.3.1 Implementation

In GMM, we use expectation maximization. In this question, we have tried learning using our training data with different Gaussian mixtures. We have 5 output classes and a 10-dimensional feature vector. We have randomly initialized our means.

3.3.2 Results

We ran for algorithm for a sufficient number of epochs and got an accuracy of around 57%.

Gaussian mixtures.	Accuracy
1	56.66
2	56.88
3	56.96
4	58.87
6	56.86
5	56.96

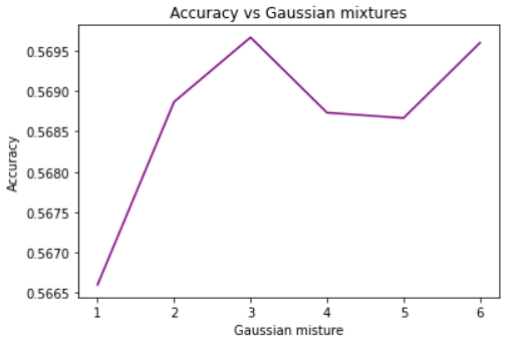

Figure 3. Accuracy vs Gaussian mixtures for question-3.

3.3.3 Observations

We tried our model with several epochs and for several Gaussian mixtures but we could not improve the accuracy of our model above 60%. There could be multiple reasons for this:

- Observed training data is not reliable or it may contain many outliers which did not allow us to learn the true distribution.
- Our model is settling for some local extrema rather than global extrema. This can be a problem because GMM's performance is dependent upon how we initialize our initial means and variance. There are many works has been done to get better initialization for GMM.
- We have plotted the likelihood vs the number of Gaussian mixtures curve and we observed that we get different curves for different initialization of means. This also indicates that GMM is initialization dependent and is correct.
- We also observed that for a particular class of data misclassification into different classes is done in the almost same ratio for all the classes and for all the Gaussian mixtures we have observed. This can be if some noise is added in all the classes in equal proportions.
- We are getting the almost same value of F1 score for every class. This can happen if the distributions from which different classes are coming are all symmetric in space and there is an almost equal and fairly large overlap region where these distribution overlap. This may also justify the low accuracy.
- We also observe that accuracy and likelihood first increase as we increase the Gaussian mixtures but after a time it slightly decreases and then again increases. This could be due to the random initialization of means

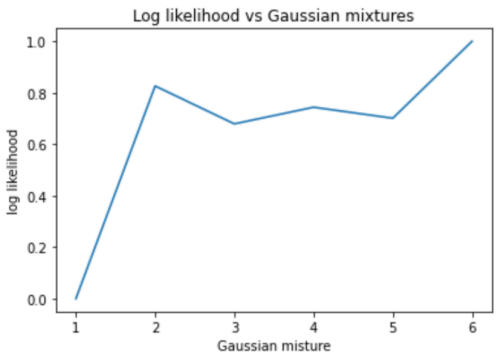

Figure 4. Log likelihood vs Gaussian mixtures for question-3.

and variance. The other reason could be that the number of mixtures we are considering is not matching with the actual distribution but eventually, it increases with the increasing number of Gaussian mixtures.

3.4. Multi class logistic regressor

3.4.1 Implementation

- Here softmax function is used as conditional probability of a class and cross entropy loss is used as loss function.

$$g_j(W,X) = \frac{e^{W_j^T.X}}{\sum_s e^{W_j^T.X}} \tag{5}$$

$$L = -\sum_{i=1}^{n}\sum_{j=1}^{k} y_i^j ln(g_j(W,X_i)) \tag{6}$$

- For every step the following update rule is followed:

$$W_j(k+1) = W_j(k) + \alpha \sum_{i=1}^{n} (y_i^j - g_j(W,X_i))X_i \tag{7}$$

- Two functions are created to implement the logistic regressor. 'train_epoch1' function and 'calc_loss' function.

- 'train_epoch1' function is called for each epoch we want to run the algorithm. It sends each data element from train set (i^{ith} row) to 'calc_loss' function which calculates loss for that particular data element and gradient for each of the W's for that particular data element and returns it back to 'train_epoch1' function which adds up the loss and gradient for each of data elements(whole dataset) and returns back so that we can update our parameters.

- Then different metrices for testing datasets were found.

3.4.2 Results and Observations

The following metrices are calculated:

- EMPIRICAL RISK: The model was trained for 25 epochs as the loss decrease rate was low after 15 epochs and at the end of 25 epochs the empirical risk on training data was found to be 1.15 .

  The empirical risk on test data was found to be 1.14

- ACCURACY : The accuracy on test data was found to be 57.32%

- F1 SCORES: The F1 scores of corresponding classes are: class 1 :0.52, class 2 :0.56, class 3 :0.62, class 4 :0.58, class 5 : 0.58

- CONFUSION MATRIX: Fig 5 shows the confusion matrix on test dataset.

```
[[1515.  391.  298.  376.  391.]
 [ 374. 1698.  298.  286.  325.]
 [ 307.  306. 1881.  257.  289.]
 [ 339.  345.  262. 1739.  290.]
 [ 350.  327.  253.  338. 1765.]]
```

Figure 5

3.5. Parzen Window

3.5.1 Implementation

The kernel function considered here is a gaussian kernel, for each test point given, the class conditional density is calculated by summing the value of gaussian kernel for each train data point of given class and the maximum density value class is chosen as the predicted class.

3.5.2 Results and Observations

The following metrices are calculated:

- ACCURACY : The accuracy on test data was found to be 57.46%

- F1 SCORES: The F1 scores of corresponding classes are: class 1: 0.52, class 2: 0.55, class 3: 0.62, class 4: 0.59, class 5: 0.57

3.6. Linear Classifier using One vs rest

3.6.1 Implementation

At first the datas are separated class wise. For each class a linear discriminant function $W_j^T X$ is defined that is trained to predict if the data point X belongs to that class or not. The equation 1 is used to find the optimal value of each W_j. The discriminant functions are run over test dataset and the first class where it is classified ($W_j^T X > 0$) is considered the predicted class.

3.6.2 Results and Observations

The accuracy is not that good for this model (around 22percent) in comparision to other models, probably because the classes are not linearly separable and hence cannot be separated using linear discriminant functions.

3.7. ROC_curve

3.7.1 Implementation

- Any 2 classes are choosen and a new dataset containing these 2 classes are created. Now simmilar to bayes classifier class conditional density of these 2 classes are calculated and their difference is compared to thresholds. As the threshold is increased from minimum set value to a maximum set value the graph is plotted

- We also tried another method of plotting ROC curve by considering any two classes and any one feature from among 10 features.

3.7.2 Results and Observations

The second method of ROC gave better curve compared to first one (Fig 6). In second method only a few features from 10 features gave a nice curve, some feature gave a straight line (the worst ROC curve is a straight line) which implies they are not good feature to classify our class on, while some features that gave good results can be considered as good features to classify our classes.

3.8. Problems Faced

One of the problem faced (in multiple logistic regression) was the alpha(step size parameter) needs to be changed as time progress else the error first decreases then increases because of large step size. Hence to mitigate this issue a conditional statement was added that decreased the alpha parameter by a factor of 10 the moment current epoch's error becomes greater than previous epochs error.

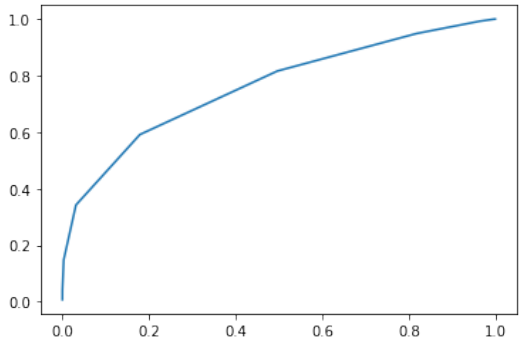

Figure 6. ROC curve

4. Question 4

Here we have been given an image dataset called Kannada MNIST containing digits that we have to classify into 10 classes.

At first the zip file containing all the images are uploaded to google colab and unzipped, then each image is converted to a feature vector by flattening it and then stored in a dataset. Further the dataset is divided into train and test dataset according to a given ratio. For importing each image to colab 'os' library is loaded into colab and to read images into numpy array cv2 image library is loaded ('cv2' and 'os'library only used for loading image and converting into numpy array). Each feature is normalised by dividing with 255 (as image value ranges between 0-255). Various models like bayes, logistic regressor are created and different metrices are computed.

Model 1 is 20:80 split. Model 2 is 30:70 split. Model 3 is 50:50 split. Model 4 is 70:30 split. Model 5 is 90:10 split.

Confusion matrix and F1 scores are printed in ipynb file but not shown here due to space constraints, please do refer to ipynb file for confusion matrix and F1 scores of each model.

4.1. Naive Bayes Classifier

4.1.1 Implementation

- The datas are separated into different classes and then their means were calculated by taking average over all classes (because of naive bayes assumption). Variance is assumed '1' for each feature

- ML estimate for mean(m_j) for each class (C_j)is

$$m_j = \frac{\sum_{\forall i \in C_j} X_i}{n} \tag{8}$$

- For every data point class conditionals are calculated for each class using the below formulae.

$$g_j = \frac{1}{\sqrt{2\pi}} e^{-\frac{||x-\mu||_2^2}{2}} \tag{9}$$

- The class where the class conditionals is highest is choosen as the predicted class.

4.1.2 Results

Naive bayes' Outputs for different models

Model no.	Train Accuracy	Test Accuracy
1	83.74	84.36
2	84.25	84.37
3	84.82	86.12
4	85.09	85.89
5	85.88	81.7

4.1.3 Observations

In naive bayes case the assumed variance is '1', We have also tried using updating variace and mean both but the accuracy we got was nearly same as that of choosing variance = '1'. Thus in the final code we decided to retain only the simple model by fixing variance as '1'. The train and test accuracy are high and nearly same which implies they operate in optimum region of low bias and low variance. As the number of training data points are increased accuracy increases (as expected because I have now a larger dataset to train on) but surprisingly accuracy falls in last model of 90:10 split. This happens probably because the data is trying to overfit (for naive bayes model) as we have very large number of training examples compared to test.

4.2. Logistic regressor

4.2.1 Implementation

- The dataset was divided into a choosen ratio.
- Here softmax function is used as conditional probability of a class and cross entropy loss is used as loss function. Equations is same as eq-5,6 and 7.
- The same two functions are used that was created in P3 logistic regressor regressor.
- Here it was observed that after around 40 epochs the decrease in loss was very low hence number of epochs for training for each model is taken to be 50.
- Then different metrices for testing datasets were found.

4.2.2 Results

Table 1 : Logistic regressor Outputs for different models

Model no.	Train Risk	Test Risk	Test Accuracy
1	0.77	0.75	86.13
2	0.73	0.72	87.03
3	0.77	0.71	87.68
4	0.76	0.71	87.85
5	0.71	0.79	84.38

4.2.3 Observations

In logistic case also accuracy increases with increase in train data except for the last model(90:10 split) where accuracy decreases because of overfit. Train empirical risk shows no visible variation but test empirical risk decreases slowly except in the last case of 90:10 split where it increases probably because the model is trying to overfit due to large train set in comparison to test dataset. Another reason of overfit can be the large dimension feature (784).

It is like the bias-variance curve, with increase in training data the accuracy of train dataset increases and train error decreases. But for test dataset accuracy first increases and then decreases after certain point when it crosses from underfit region to overfit region. That is why test loss first decreases and then increases suddenly.

4.3. GMM

4.3.1 Implementation

In this question, we have an image dataset. We first loaded the images as a NumPy array, the size of each image is 28*28, and then we flattened it into a 784-dimensional vector. In this question, we have tried learning using our training data with different Gaussian mixtures. We also tried splitting the data into training and testing with different ratios, for example in 20:80, 30:70, 50:50, 70:30, and 90:10. We have 10 output classes and a 10-dimensional feature vector. We have randomly initialized our means.

4.3.2 Results

We ran for algorithm for a sufficient number of epochs. Accuracy for different sets of parameters is given below.

GM	20:80	30:70	50:50	70:30	90:10
1	85.57	85.21	85.84	84.97	86.03
2	87.88	88.15	88.61	86.45	89.11
3	89.34	89.06	89.62	89.03	89.61
4	89.16	89.45	89.41	89.45	90.03
6	89.70	89.60	90.13	89.41	90.01
5	89.52	89.46	90.31	89.41	90.53

GM refers to Gaussian models

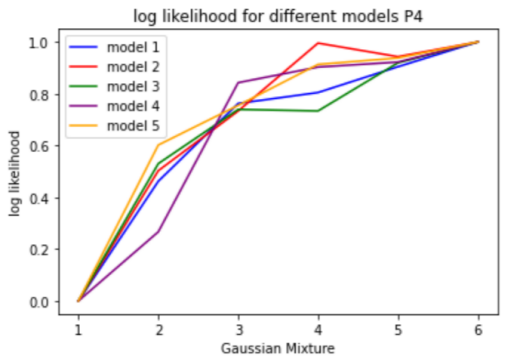

Figure 7. Log likelihood vs Gaussian mixtures for question-4.

4.3.3 Observations

In this question, we have a high-dimensional feature vector of size 784. We know that as the dimension increases there are many problems that arise. Some of them are observed by us.

- We got very good test accuracy in all the cases in this question. Sometimes accuracy can be a bad measure of model performance, although in our case we have balanced data from all classes for testing so accuracy indeed represents how good the model is in our case.
- We also calculated the F1 score for all the cases and it is in the range of 0.89-0.98, which is a fairly good range.
- It was observed that as we increase the size of the training, we get good accuracy on test data, this is intuitive because more training data means we have large samples from the actual distribution thus we get a better prediction for the actual distribution.
- Although it seems that as the size of the training dataset is increasing test accuracy is also increasing but sometimes it is dropped by some fraction. The reason for this could be the random initialization of means. Because if we choose the wrong initial means algorithm may take a large number of epochs to converge.
- As we discussed earlier, it takes time to compute a large number of epochs.

4.4. ROC_curve

4.4.1 Implementation

Any 2 classes are choosen and a new dataset containing these 2 classes are created. Now simmilar to bayes classifier density of these 2 classes are calculated and their difference is compared to thresholds. As the threshold is increased from minimum set value to a maximum set value the graph is plotted for each model.

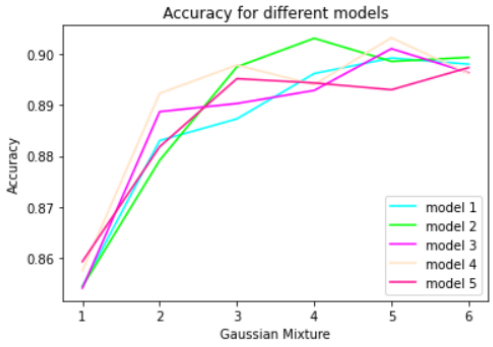

Figure 8. Accuracy vs Gaussian mixtures for question-4.

4.4.2 Results

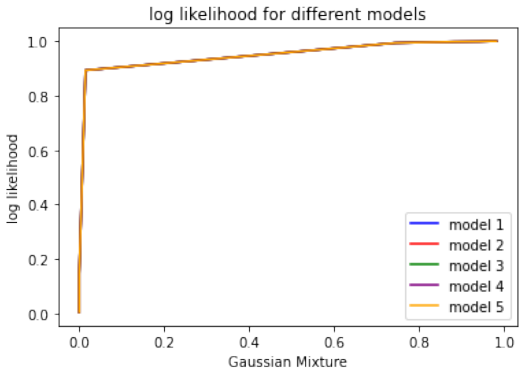

Figure 9. ROC curves for different models

4.5. Problems Faced

If features are not normalised then we get erroneous results (in naive bayes and logistic case) , because if values are high then exponent of large negative terms becomes zero for all the terms and hence it throws error. Hence to prevent this the features are normalised by dividing with 255.

Another problem faced was the loss after some epochs (in logistic regression case) starts increasing and after debugging it was found to be the fault of step size. Thus we used a variable step size that decrease itself by a factor of 10 when current loss is more than previous epoch's loss.

We faced several challenges while dealing with the image dataset. At first, we had a very large image dataset, each image of size 480*640 which not only takes time in loading in NumPy, but in this case, we will have a huge covariance matrix for which we have to calculate the inverse. This is computationally exhaustive. Not only this we need a huge

number (307,200) of independent data points otherwise our covariance matrix will not be invertible.

Even if have 307,200 images for each class, we will still get our covariance matrix non-invertible because many images will be almost the same, and again we face the same problem with the invertibility of the covariance matrix. So we need a huge dataset in this case.

So we reduced the size of the image to 28*28 and then flattened it to get a 784-dimensional vector. Although we still faced the problem of the invertibility of the covariance matrix.

While calculating the Gaussian density, we need to multiply it with constant $\frac{1}{(2\pi)^{\frac{dimension}{2}}}$, but we have a huge dimension of feature vector because of which this term becomes almost 0 and thus makes everything 0.

5. Question 5

Here the images from previous datasets are condensed to 10 features using PCA and we need to classify these digits into their corresponding classes.

The PCA dataset for this question is loaded into colab and datas are separated into train and test datas and various models like bayes, logistic regressor are created and different metrices are computed.

Model 1 is 20:80 split. Model 2 is 30:70 split. Model 3 is 50:50 split. Model 4 is 70:30 split. Model 5 is 90:10 split.

Confusion matrix and F1 scores are printed in ipynb file but not shown here due to space constraints, please do refer to ipynb file for confusion matrix and F1 scores of each model.

5.1. Naive Bayes Classifier

5.1.1 Implementation

Same procedure is used for building naive bayes model as described in section 4.1.1

5.1.2 Results

Naive bayes' Outputs for different models

Model no.	Train Accuracy	Test Accuracy
1	82.45	82.23
2	81.71	82.96
3	81.72	83.79
4	82.24	85.76
5	82.98	86.46

5.1.3 Observation

Here also variance assumed is '1' as in previous question case. The train and test accuracy are high and nearly same which implies they operate in optimum region of low bias and low variance. Here the accuracy increases with increase in training dataset and as number of features are small there is no overfit and hence unlike the previous question accuracy increases even in 90:10 split case.

5.2. Logistic regressor

5.2.1 Implementation

The same process is followed as mentioned in section 4.2.2 except here the number of epochs taken is 100 instead of 50 as it was observed the decrease in loss still goes on after 50 iterations.

5.2.2 Results

Logistic regressor Outputs for different models

Model no.	Train Risk	Test Risk	Test Accuracy
1	0.68	0.67	83.89
2	0.70	0.67	84.69
3	0.71	0.63	86.14
4	0.69	0.58	87.25
5	0.66	0.57	88.11

Table 2 : Outputs for different metrices and models for Logistic regression

5.2.3 Observation

In logistic case also accuracy increases with increase in train data and test empirical risk decreases with increase in train data. Here as the feature dimension is less hence there is no overfit of the model and thus unlike the previous question our accuracy increases even in the 90:10 split.

5.3. GMM

5.3.1 Implementation

This question is quite similar to the previous question. In the previous question, we were given the images for different numbers but in this question, we are given a 10-dimensional feature vector in place of the actual image. This can be achieved using various techniques for example PCA can be used for this.

In this question, we also split the dataset into training and testing in different ratios like in the previous question.

5.3.2 Outputs

We ran for algorithm for a sufficient number of epochs. Accuracy for different sets of parameters is given below.

GM	20:80	30:70	50:50	70:30	90:10
1	93.97	94.01	94.17	93.93	93.88
2	94.74	94.53	94.86	94.53	94.73
3	95.07	94.99	95.25	94.88	95.00
4	95.29	95.16	95.39	95.11	95.06
6	95.19	95.40	95.46	95.22	95.21
5	95.41	95.19	95.39	95.16	95.42

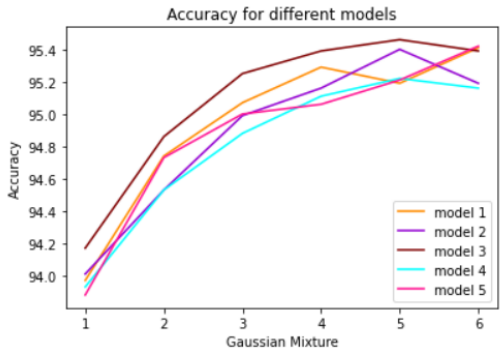

Figure 10. Accuracy vs Gaussian mixtures for question-5.

5.3.3 Observations

We have observed that it was easier to work with the data given for this question than the data given for the previous question because of the issues faced due to the high dimensionality of the feature vector.

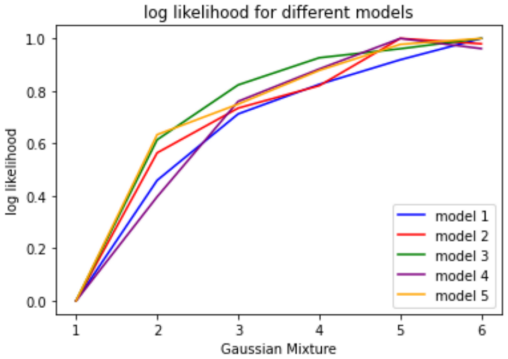

Figure 11. Log likelihood vs Gaussian mixtures for question-5.

- In this question, we got good accuracy for all the cases.
- For some cases, we got accuracy up to 96%, this indicates that our training data was clean and is actually true representative of the actual distribution.
- Although we get to see the trend like previous that as we increase the size of the training dataset accuracy increases but there can small amount of drop sometime that is again due to random initialization.
- We have run our Colab notebook and the results can be seen there but it is worth noting that if run the code again we may not get the exact same result and the reason is again random initialization of means and covariances. But we will get results that are close to the current ones.

5.4. ROC_curve

The implementation is simmilar as described in P4.

5.4.1 Results

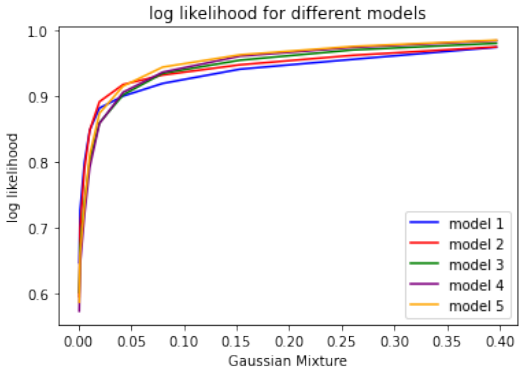

Figure 12. ROC curve for different models.

As shown the TPR and FPR varies nicely with increase in threshold. This suggests that we can compare the difference of class conditionals with a threshold and decide on class and chosing the optimal threshold value will give us a high accuracy.