EECE 5644: Machine Learning Project 3 Report

1. **MLP**: Given the following data:

$$Priors = [0.25, 0.25, 0.25, 0.25]$$

$$number\ of\ samples = [100, 200, 500, 1000, 2000, 5000, 100000]$$

$$m_0 = \begin{bmatrix} 20 \\ 0 \\ 0 \end{bmatrix} C_0 = \begin{bmatrix} 10 & 0 & 10 \\ 5 & 40 & 0 \\ 5 & 0 & 20 \end{bmatrix} m_1 = \begin{bmatrix} 0 \\ 5 \\ 10 \end{bmatrix} C_1 = \begin{bmatrix} 40 & 0 & 0 \\ 0 & 25 & 0 \\ 0 & 0 & 10 \end{bmatrix}$$

$$m_2 = \begin{bmatrix} 5 \\ 0 \\ 15 \end{bmatrix} C_2 = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 20 & 0 \\ 0 & 0 & 30 \end{bmatrix} m_3 = \begin{bmatrix} 20 \\ 25 \\ 5 \end{bmatrix} C_3 = \begin{bmatrix} 15 & 0 & 0 \\ 0 & 20 & 10 \\ 0 & 0 & 25 \end{bmatrix}$$

$$\lambda\_loss = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

Assumptions:

- As stated in the question, the data distribution was generated to achieve a minimum probability of error between $10 - 20\%$.
- while optimizing the different hyper-parameters such as the epochs and number of neurons, first I choose the number of epochs by calculating the minimum probability error for each dataset. Then, using the optimal epoch, I choose the number of perceptrons. Finally, I applied the tuned parameters for the test dataset and different performance measures such as confusion matrix , probability of error were used.
- I choose the number of epoch as the optimal value after the error is flattening, because from that point there will be an over-fitting in the training data set and will perform poorly for the test dataset. While choosing optimal number of perceptrons, as the number of perceptrons increases , we will get different local minima but when the error in the curve is close to the theoretical minimum error (which we get from the optimal classifier), we can use that as number of neuron for the hidden layer.

Implementation:

In this problem, an MLP was implemented with 2 fully connected layers (one hidden layer and one output layer). After generating the samples based on uniform priors, I have used the training datasets for choosing the hyper-parameters and later implement them in the test dataset.

The samples were generated based on the selected mean,covariance and uniform prior values, by making sure those datasets have 10-20% theoretical probabilitty of error when implemented in the MAP classifier.

I used 10-fold cross validation in order to select the optimal number of perceptrons, optimal activation function type for the hidden layer ,optimal number of epochs for training, and the better optimizer. For choosing the number of epochs and perceptrons, I used a log-space function with base 5 and 3 with a maximum limit of 600. But for some of the dataset's (higher number of samples), I limit the upper number to 250 and 125 to reduce the time complexity of the training. Activation functions such as $ReLU$,elu,$Sigmoid$ were used and compared based on the error percentage I got from the cross validation. Similarly, the optimizers such as $Adam$ and SGD were compared and the one with smaller error while plotting the number of perceptrons graph was selected for each of the training datasets.

The **10-fold cross validation** is implemented in two functions ($split\_data$ and $test\_train\_split$). The first function does the partitioning of the input training data in to 10 equal number of folds by randomly selecting the samples from the input data. So the function takes the dataset and number of folds (10), and returns the partitioned folds as a list. The later function does the assignment of the test data and train data from the output of the $split\_data$ function by assigning the fold with the current index i as test data and the rest as training data. So this function basically takes the list of partitioned folds and index of the folds, and return X_train, Y_train, X_test, $andY\_test$. The above implementation will be repeated 10 time and the number of epochs,perceptrons and activation function types were selected based on the average error from the cross validation.

Theoretical Optimal classifier: This classifier was implemented on the test dataset and was used as a reference while training the MLP model with different training datasets. I have used $0-1$ loss matrix while minimizing the risk. The probability error for the test dataset was 0.1151, and similarly the other training datasets were generated based on the probability of error range specified in the question.

weight initialization: I used $random\_uniform$ weight initializer for the training. While I was training the model to get the optimal parameter's, I saved the final weights for that specific optimal number of neuron and later used it while testing the test dataset, which can shorten the time required to find the optimal weights in the test dataset.

I Dataset 100

a Actual Data Distribution

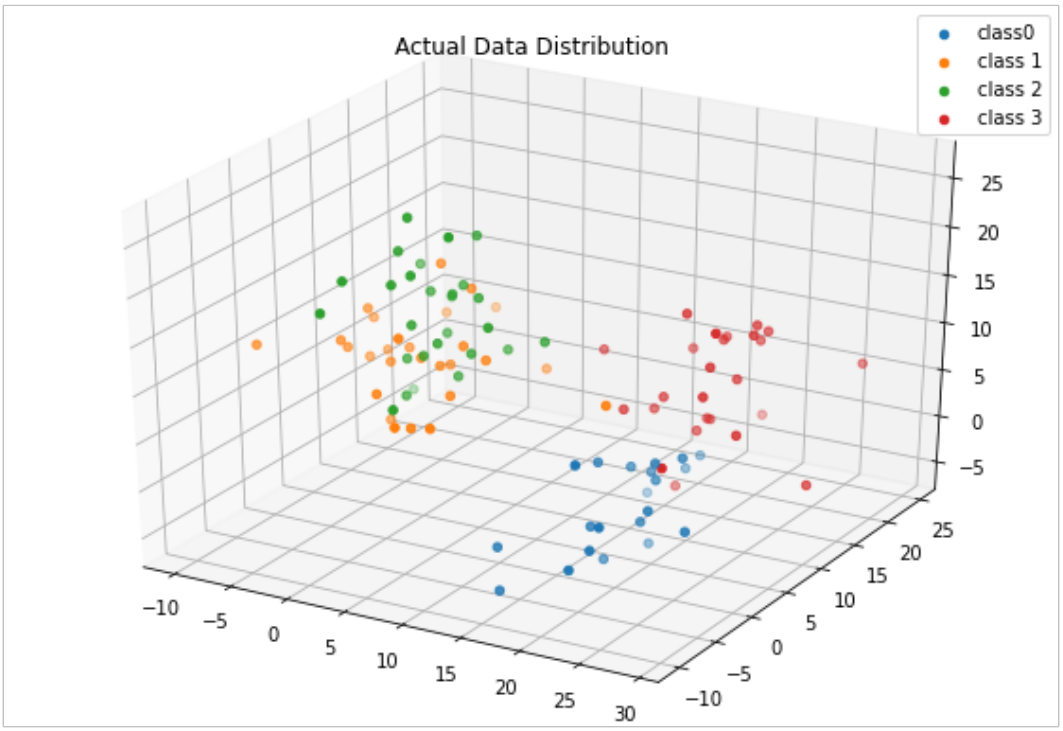

Figure 1: Actual data distribution

b Number of epochs

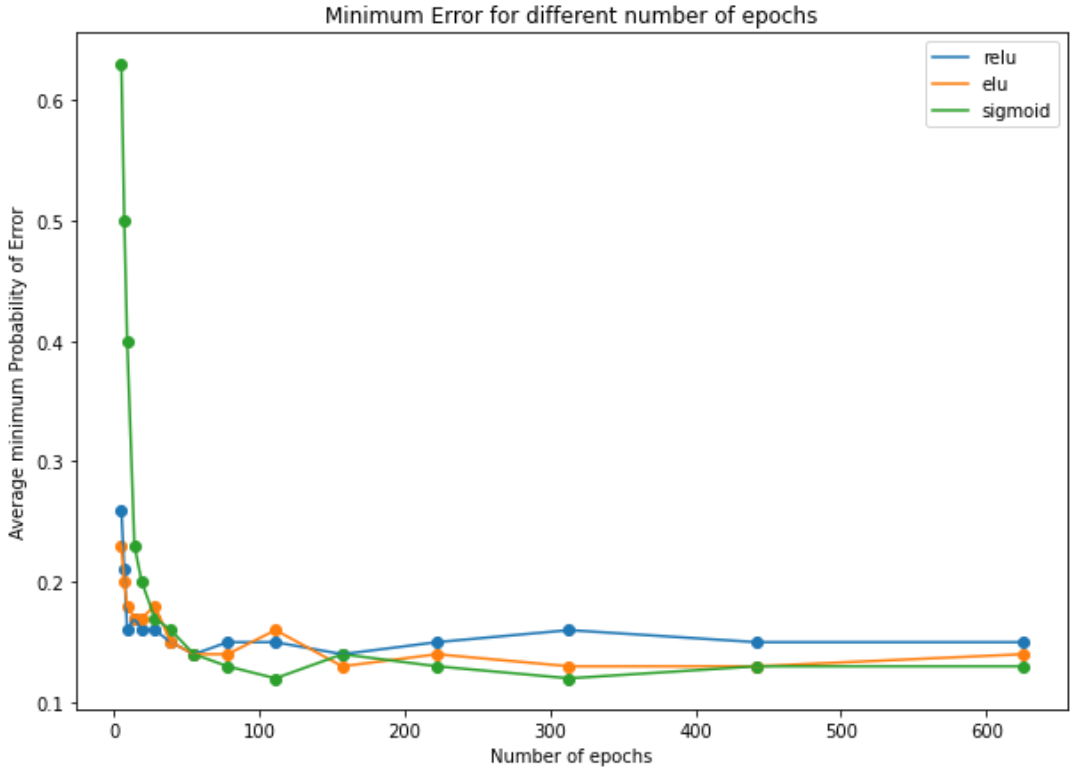

Figure 2: Minimum Error for different number of epochs

c Number of Perceptrons

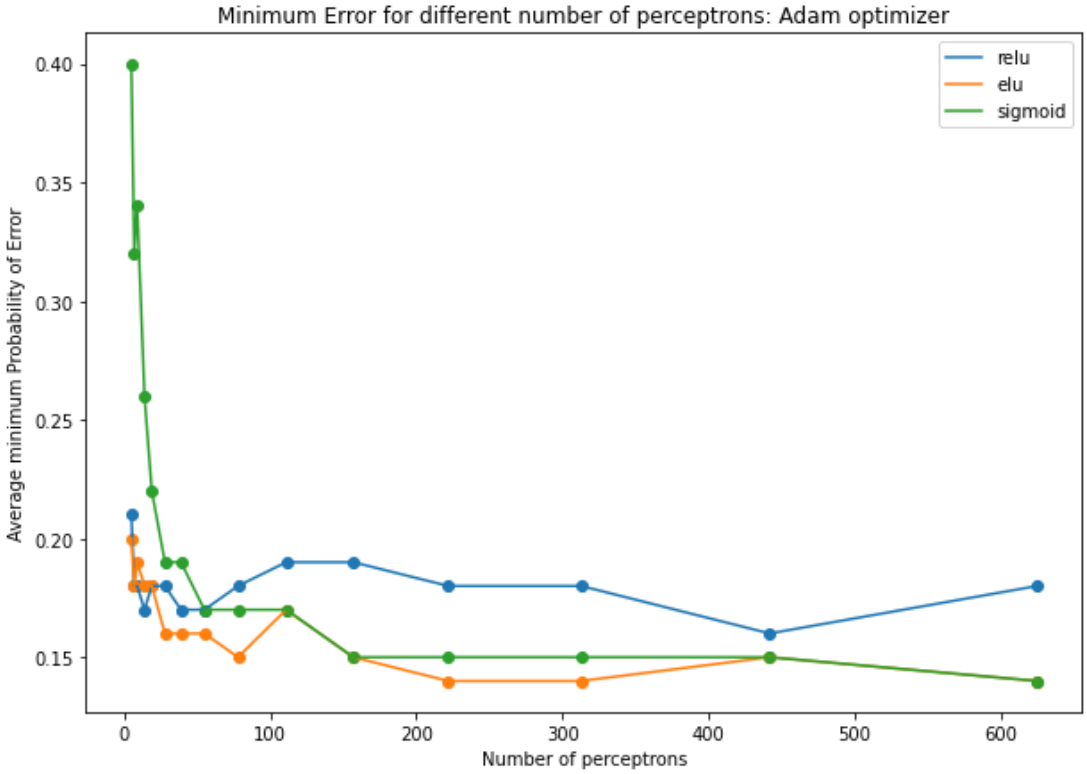

Figure 3: Minimum Error for different number of epochs under Adam optimizer

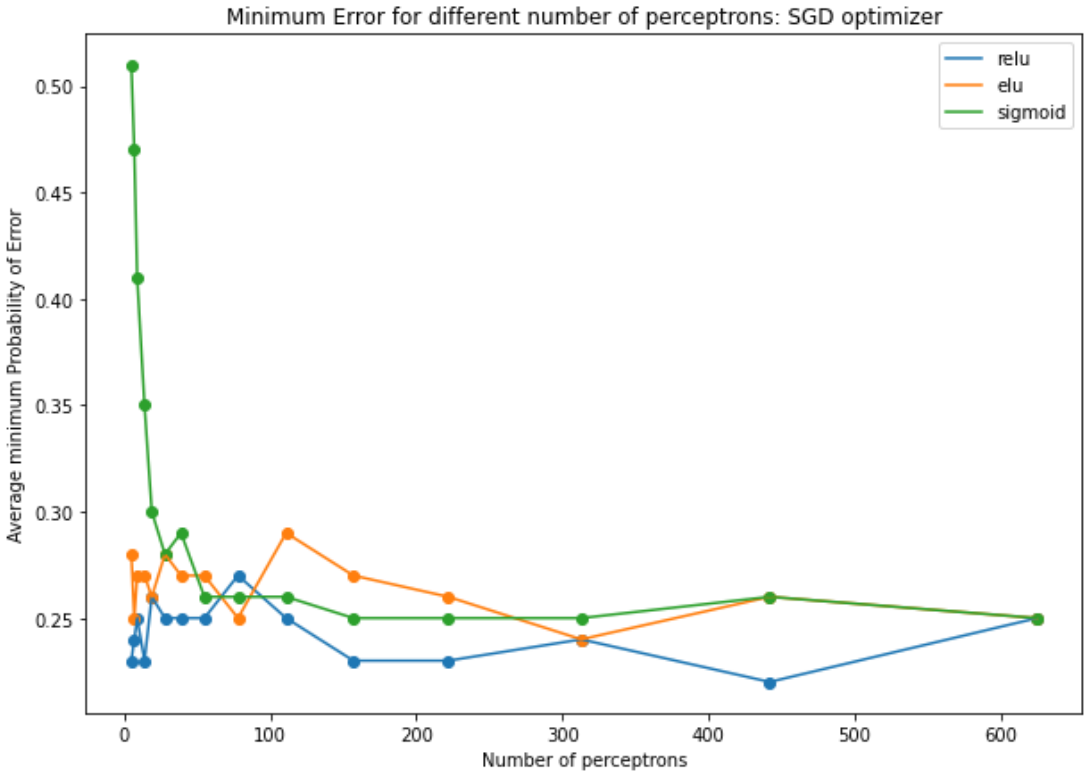

Figure 4: Minimum Error for different number of epochs under SGD optimizer

Minimum probability of error for Adam:0.14
Minimum probability of error for SGD:0.22

Based on the above plots and probability error, the following table is prepared and shows the selected parameter values for the test dataset.

Number of epochs	55
Batch size	10
Number of perceptron	150
Optimizer	Adam
Activation function for hidden layer	elu
Activation function for output layer	Softmax
loss function	categorical cross_entropy

d Confusion Matrix

The test dataset (100K), was tested based on the above parameters (in the table) given, and the confusion matrix along with the minimum probability of error is reported below. For the confusion matrix, rows are actual values while column indicates predicted values.

Confusion matrix:

$$\begin{bmatrix} 23423 & 6 & 145 & 1304 \\ 252 & 17972 & 6568 & 380 \\ 255 & 2296 & 22547 & 108 \\ 3000 & 1287 & 17 & 20440 \end{bmatrix}$$

Normalized Confusion matrix:

$$\begin{bmatrix} 0.94 & 0 & 0.01 & 0.05 \\ 0.01 & 0.71 & 0.26 & 0.02 \\ 0.01 & 0.09 & 0.89 & 0 \\ 0.12 & 0.05 & 0 & 0.83 \end{bmatrix}$$

Minimum probability of error: 0.1562
Accuracy: 84.38%
Categorical loss:0.367

II Dataset 200

a Actual Data Distribution

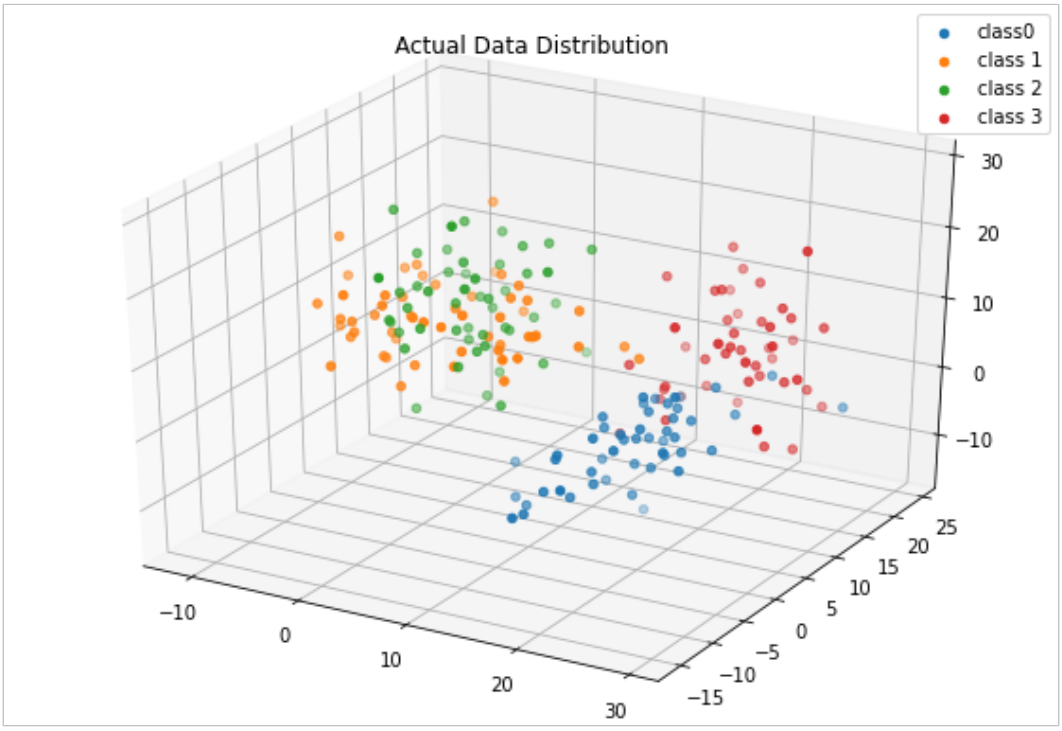

Figure 5: Actual data distribution

b Number of epochs

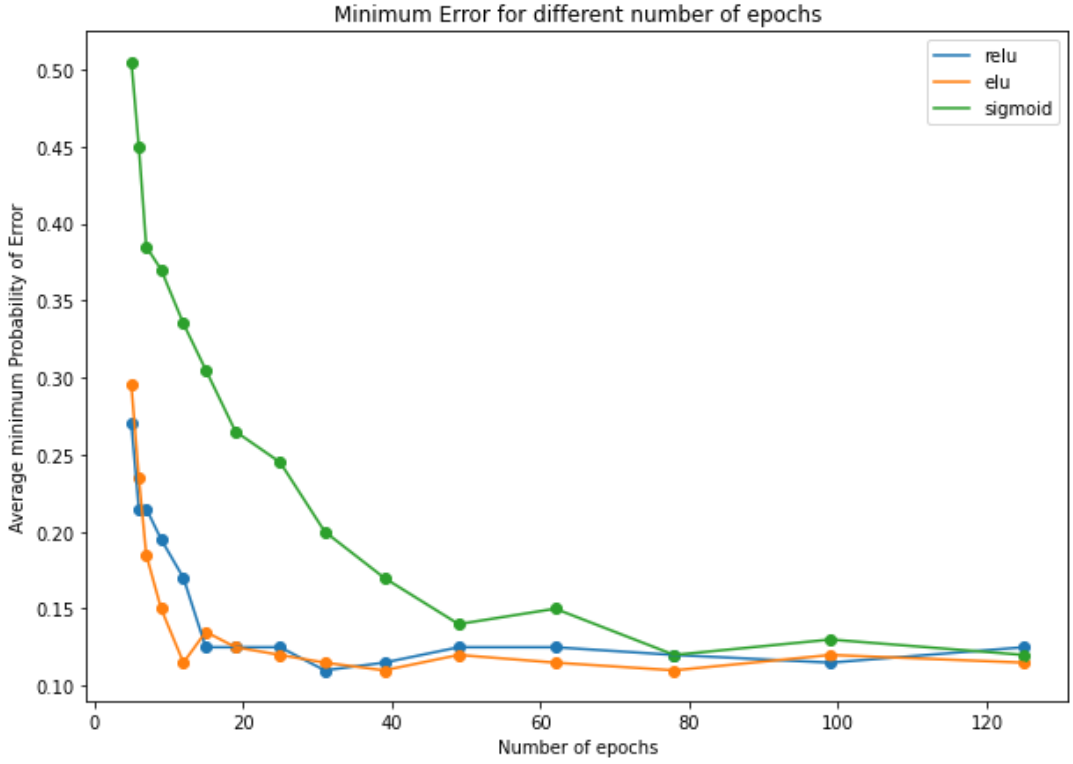

Figure 6: Minimum Error for different number of epochs

c Number of Perceptrons

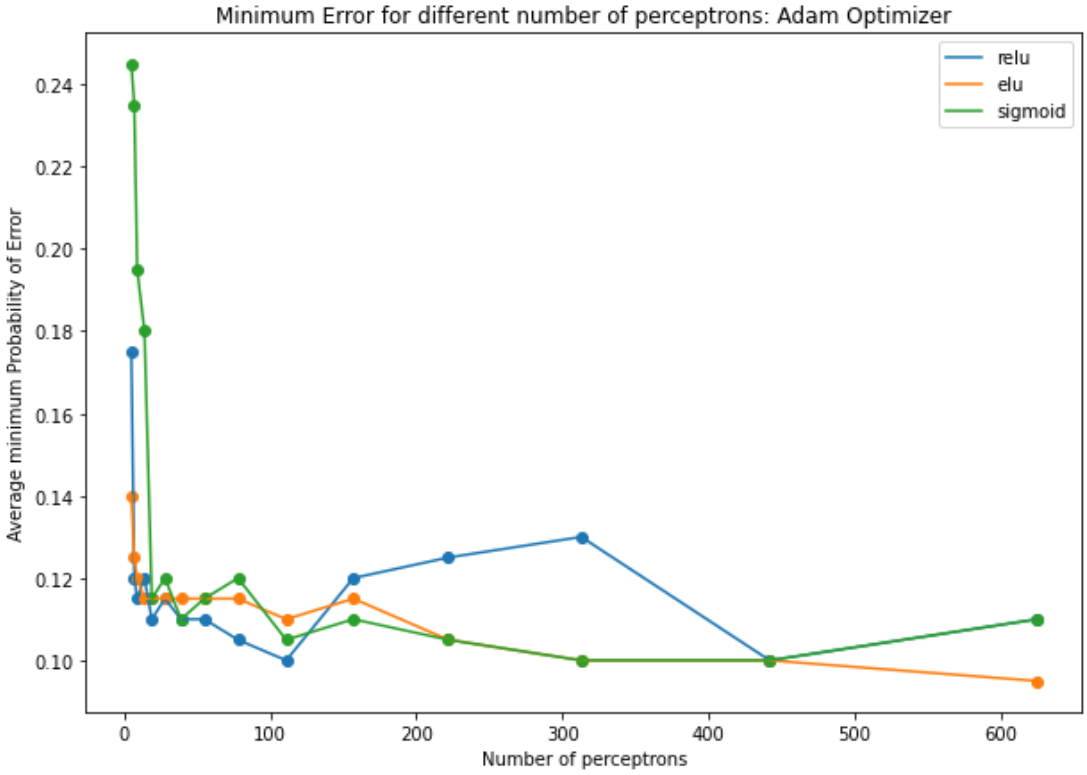

Figure 7: Minimum Error for different number of epochs under Adam optimizer

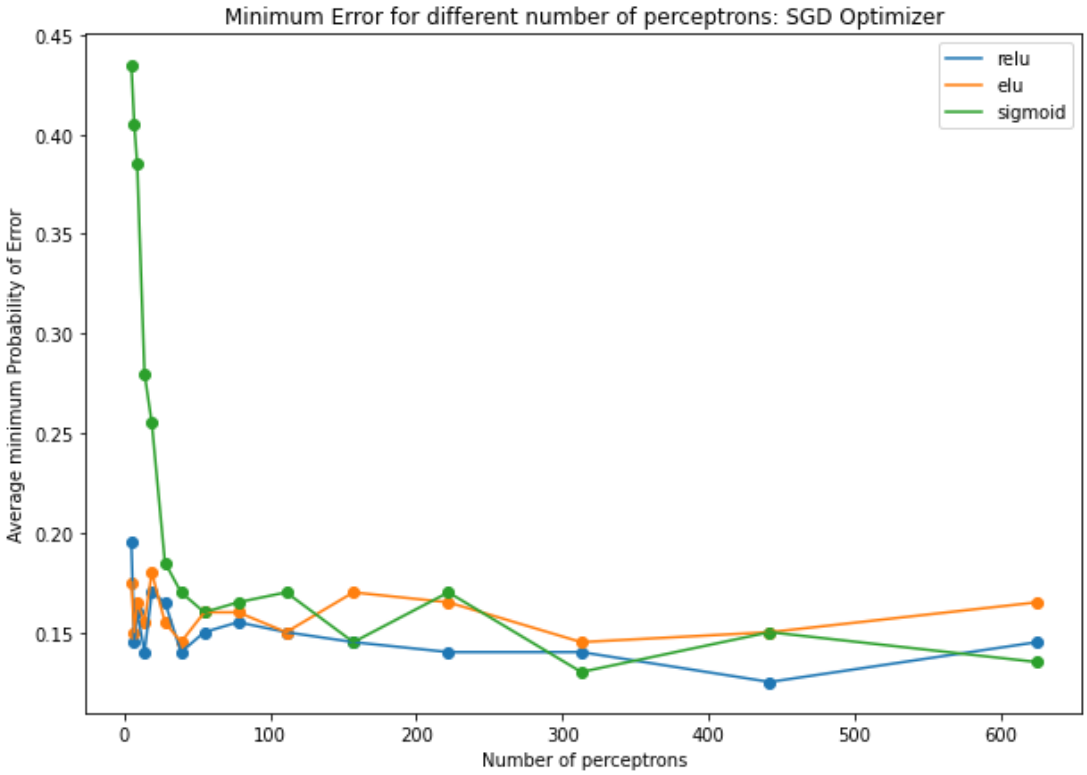

Figure 8: Minimum Error for different number of epochs under SGD optimizer

Minimum probability of error for Adam:0.1
Minimum probability of error for SGD:0.125

Based on the above plots and probability error, the following table is prepared and shows the selected parameter values for the test dataset.

Number of epoch	30
Batch size	10
Number of perceptron	78
Optimizer	Adam
Activation function for hidden layer	elu
Activation function for output layer	Softmax
loss function	categorical cross_entropy

d Confusion Matrix

The test dataset (100K), was tested based on the above parameters (in the table) given, and the confusion matrix along with the minimum probability of error is reported below. Normalized Confusion matrix:

$$\begin{bmatrix} 0.91 & 0 & 0 & 0.09 \\ 0 & 0.84 & 0.13 & 0.02 \\ 0.01 & 0.2 & 0.79 & 0 \\ 0.06 & 0.01 & 0 & 0.92 \end{bmatrix}$$

Minimum probability of error: 0.1347
Accuracy: 86.53%
Categorical loss:0.3405

III Dataset 500

a Actual Data Distribution

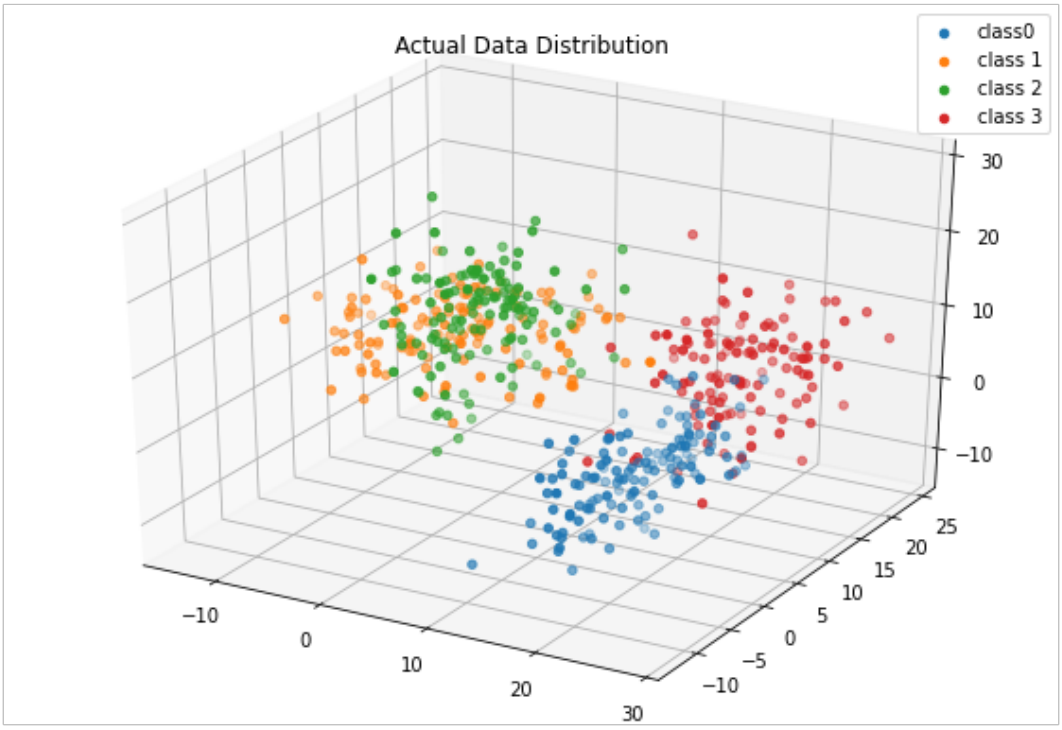

Figure 9: Actual data distribution

b Number of epochs

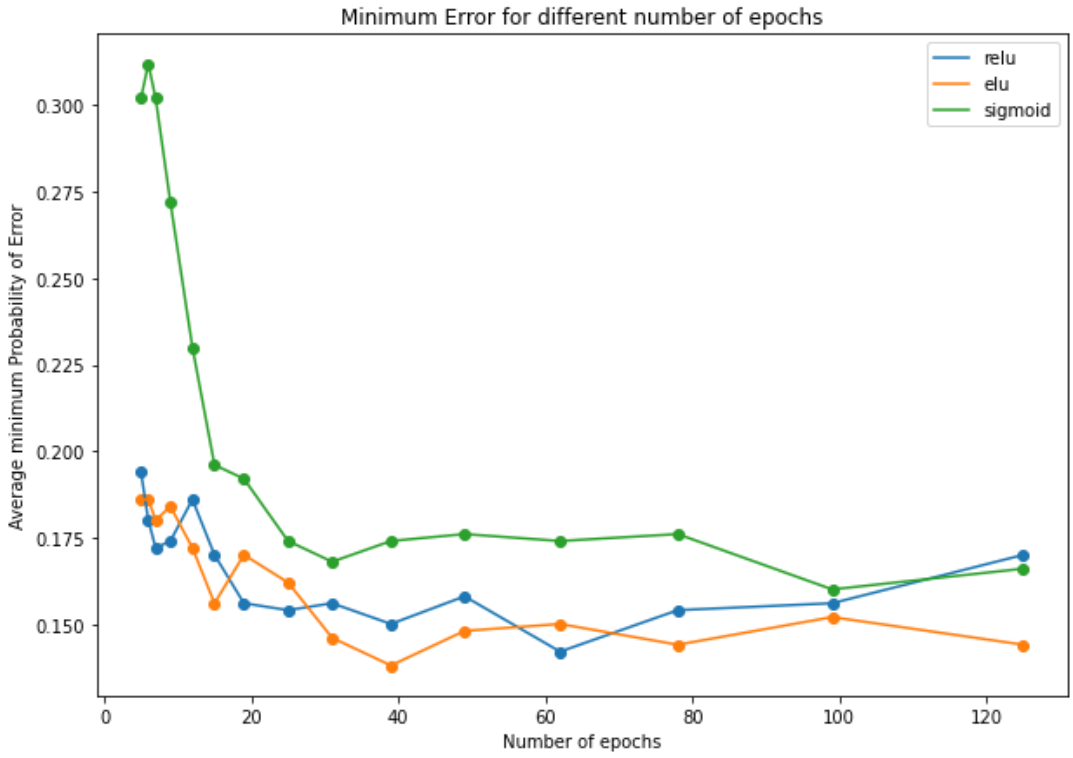

Figure 10: Minimum Error for different number of epochs

c Number of Perceptrons

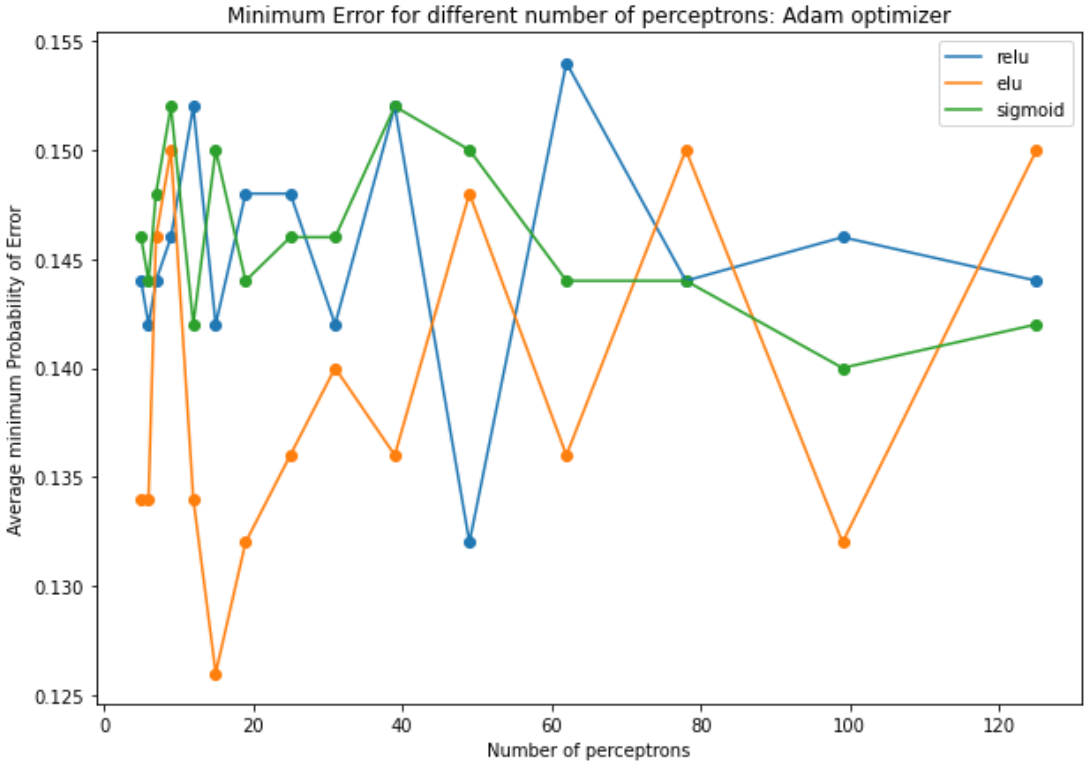

Figure 11: Minimum Error for different number of epochs under Adam optimizer

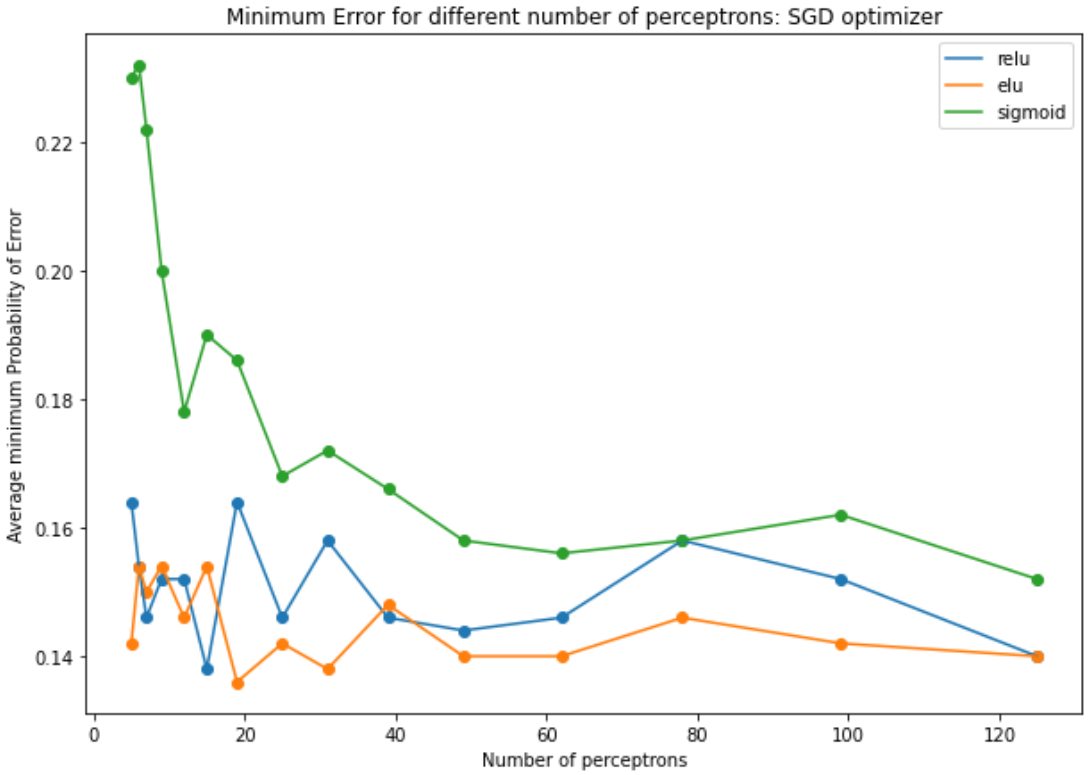

Figure 12: Minimum Error for different number of epochs under SGD optimizer

Minimum probability of error for Adam:0.126
Minimum probability of error for SGD:0.136

Based on the above plots and probability error, the following table is prepared and shows the selected parameter values for the test dataset.

Number of epochs	50
Batch size	10
Number of perceptron	15
Optimizer	Adam
Activation function for hidden layer	elu
Activation function for output layer	Softmax
loss function	categorical cross_entropy

d Confusion Matrix

The test dataset (100K), was tested based on the above parameters (in the table) given, and the confusion matrix along with the minimum probability of error is reported below. Normalized Confusion matrix:

$$\begin{bmatrix} 0.9 & 0 & 0 & 0.1 \\ 0.01 & 0.82 & 0.16 & 0.01 \\ 0.01 & 0.17 & 0.82 & 0 \\ 0.05 & 0.01 & 0 & 0.94 \end{bmatrix}$$

Minimum probability of error: 0.1308
Accuracy: 86.92%
Categorical loss:0.3099

IV Dataset 1000

a Actual Data Distribution

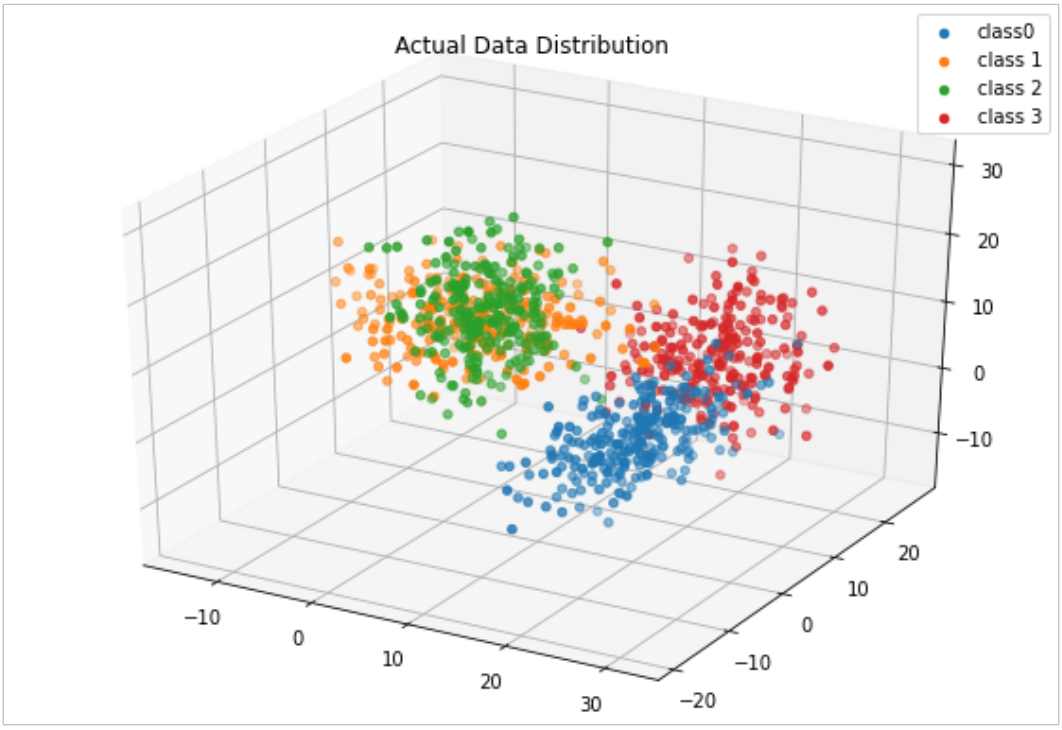

Figure 13: Actual data distribution

b Number of epochs

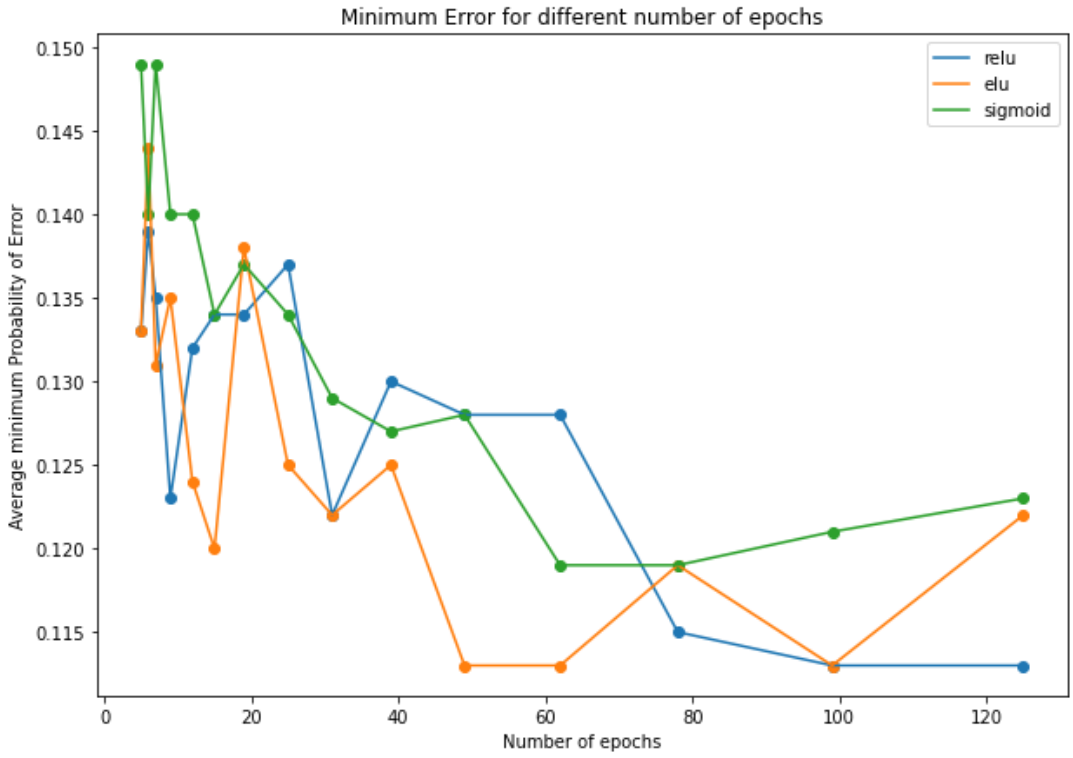

Figure 14: Minimum Error for different number of epochs

c Number of Perceptrons

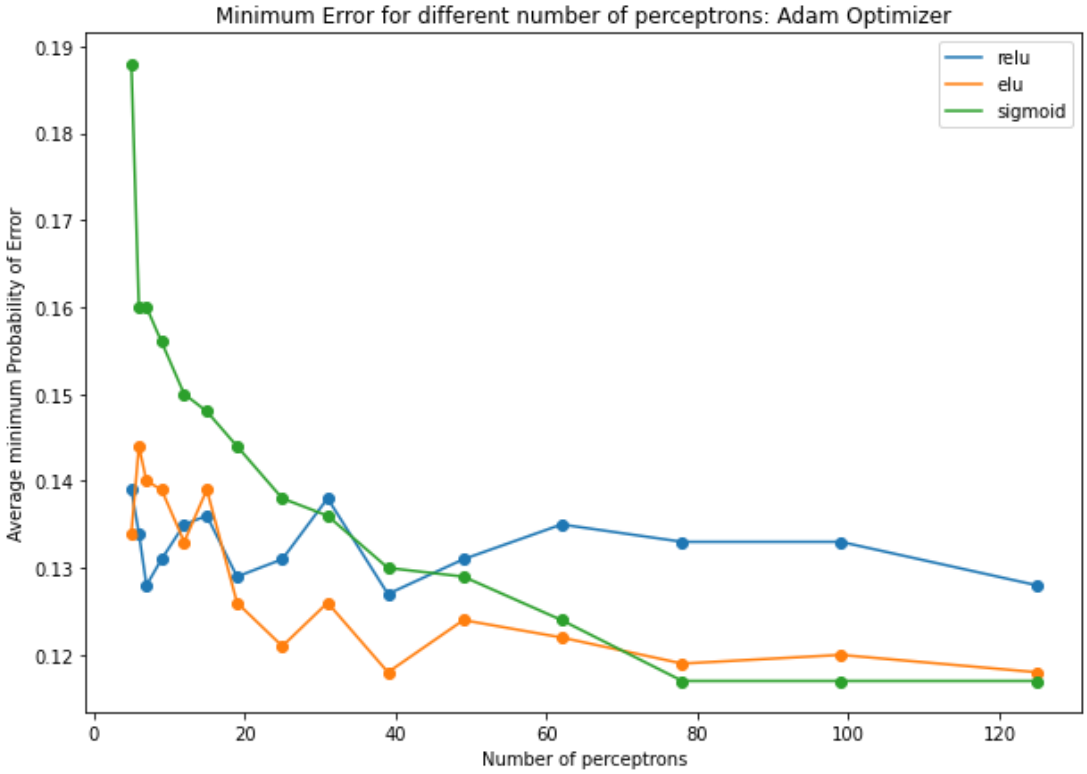

Figure 15: Minimum Error for different number of epochs under Adam optimizer

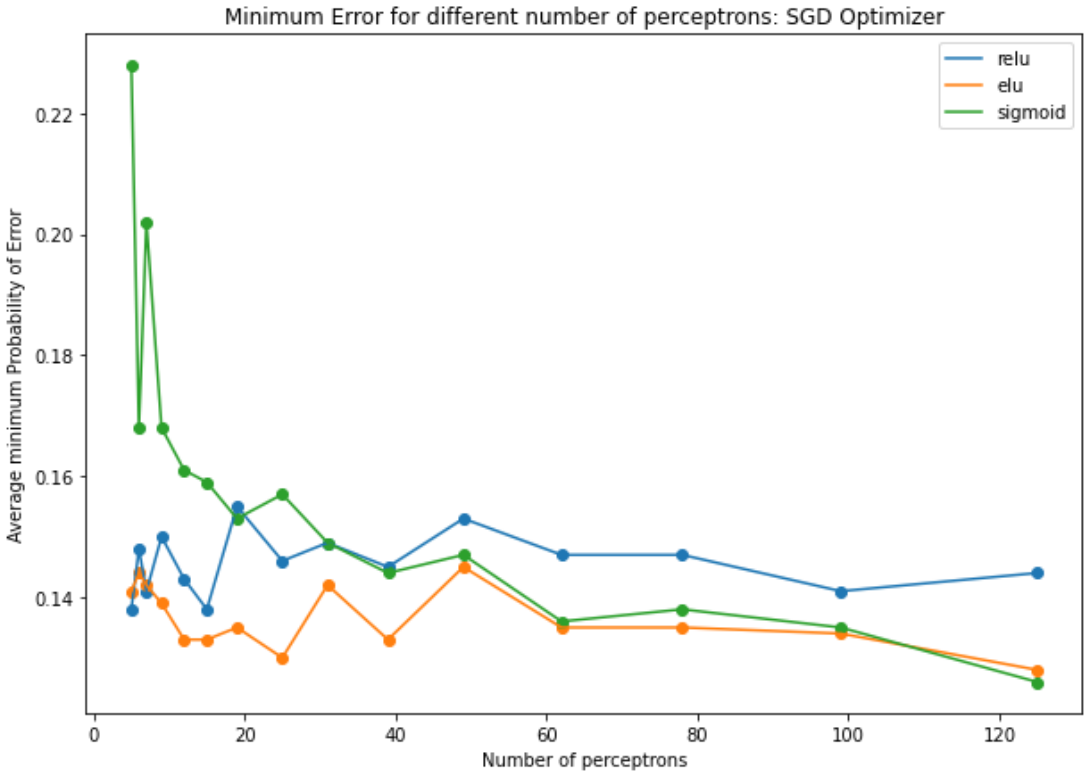

Figure 16: Minimum Error for different number of epochs under SGD optimizer

Minimum probability of error for Adam:0.117
Minimum probability of error for SGD:0.126

Based on the above plots and probability error, the following table is prepared and shows the selected parameter values for the test dataset.

Number of epochs	100
Batch size	10
Number of perceptron	78
Optimizer	Adam
Activation function for hidden layer	Sigmoid
Activation function for output layer	Softmax
loss function	categorical cross_entropy

d Confusion Matrix

The test dataset (100K), was tested based on the above parameters (in the table) given, and the confusion matrix along with the minimum probability of error is reported below. Normalized Confusion matrix:

$$\begin{bmatrix} 0.93 & 0 & 0 & 0.07 \\ 0 & 0.77 & 0.21 & 0.02 \\ 0 & 0.12 & 0.88 & 0 \\ 0.07 & 0 & 0 & 0.93 \end{bmatrix}$$

Minimum probability of error: 0.1248
Accuracy: 87.52%
Categorical loss:0.2844

V Dataset 2000

a Actual Data Distribution

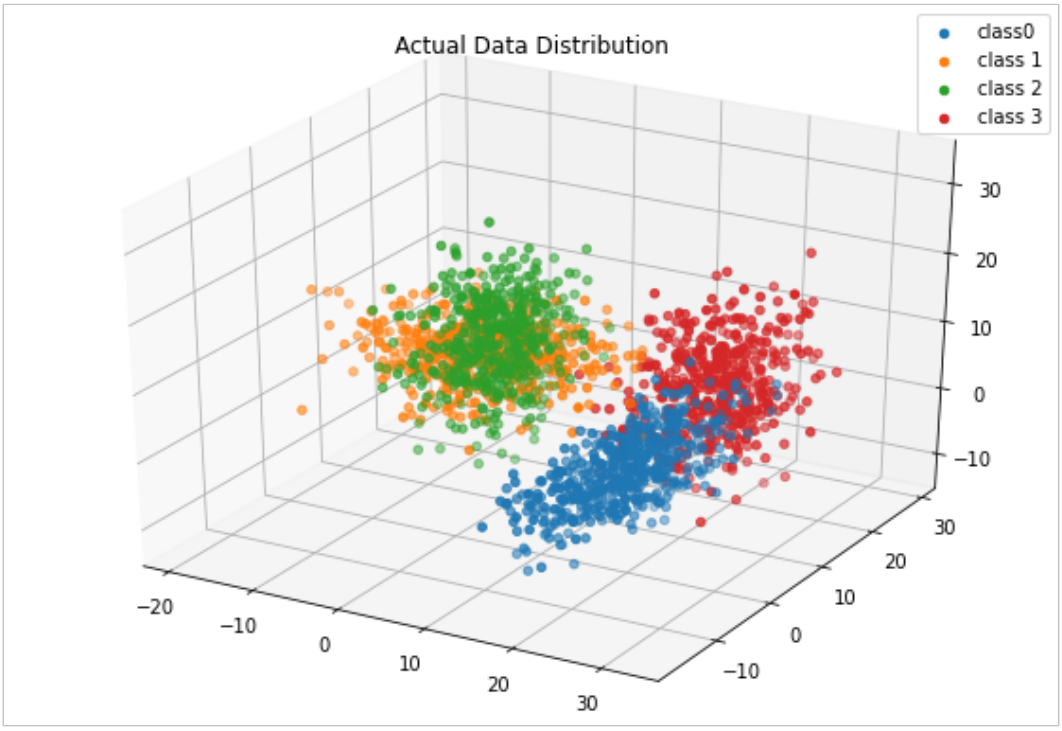

Figure 17: Actual data distribution

b Number of epochs

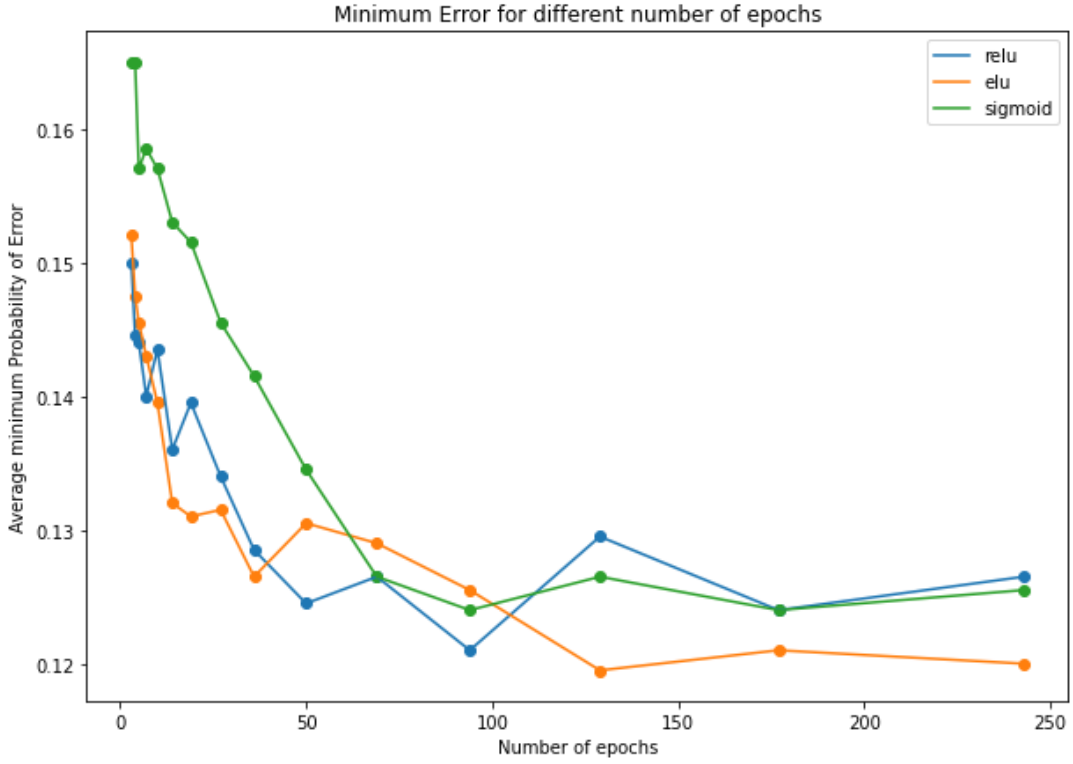

Figure 18: Minimum Error for different number of epochs

c Number of Perceptrons

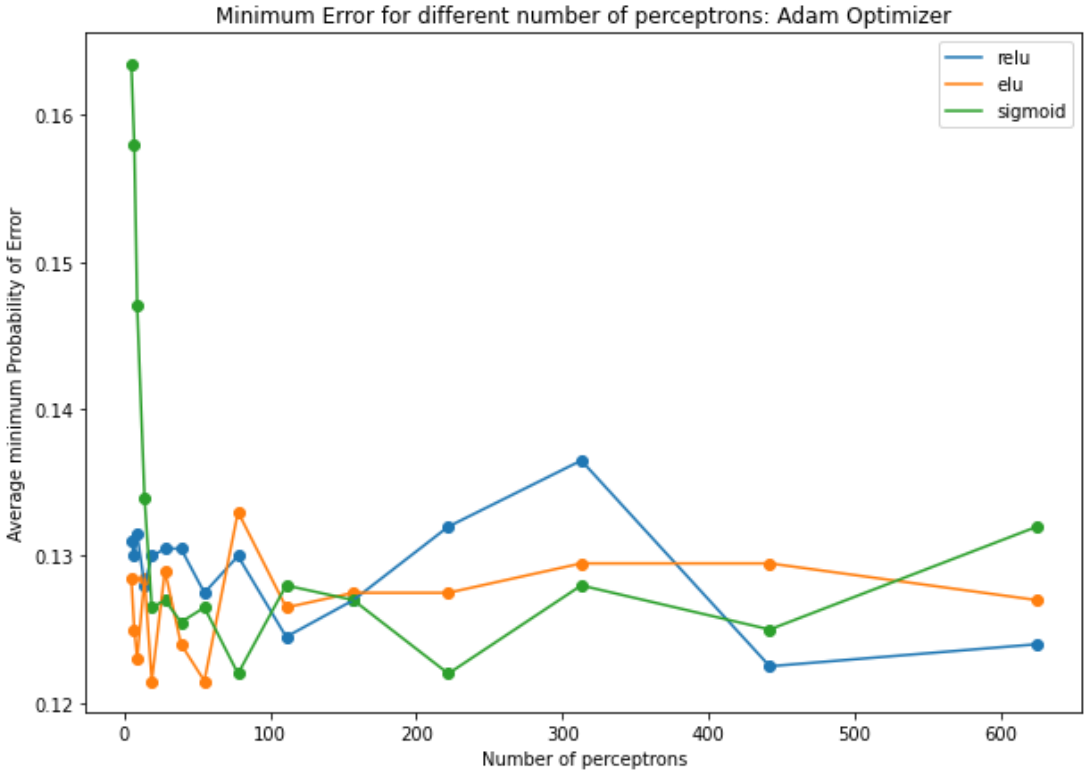

Figure 19: Minimum Error for different number of epochs under Adam optimizer

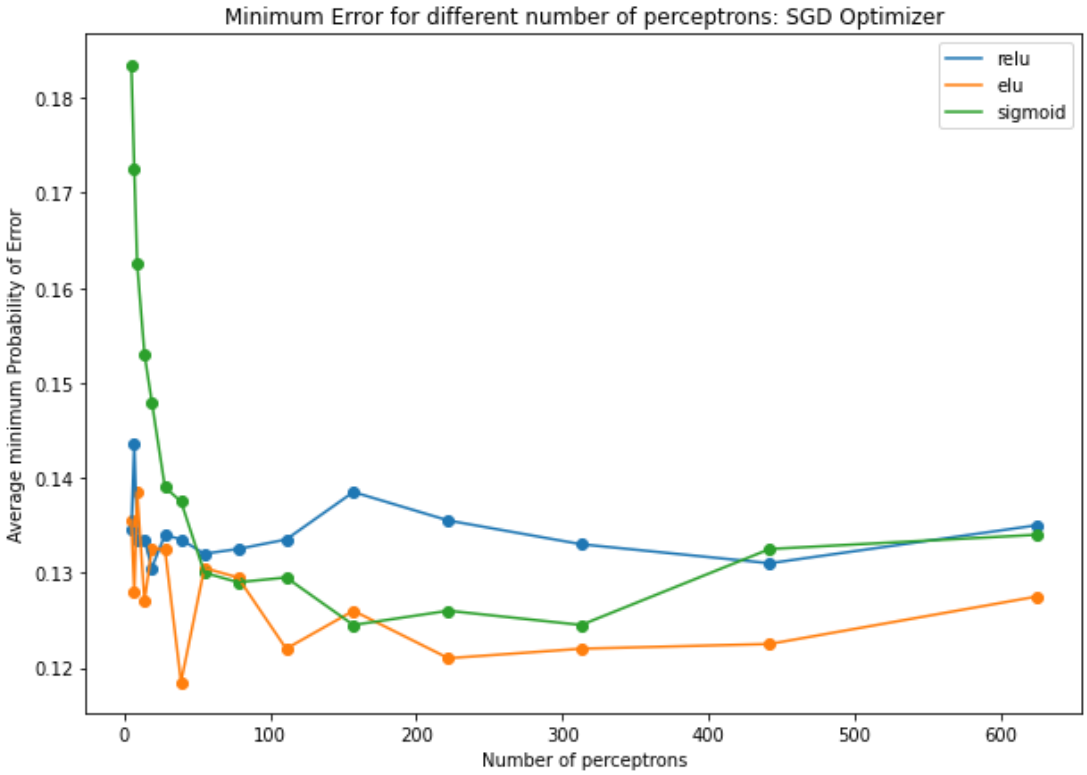

Figure 20: Minimum Error for different number of epochs under SGD optimizer

Minimum probability of error for Adam:0.1215
Minimum probability of error for SGD:0.1185

Based on the above plots and probability error, the following table is prepared and shows the selected parameter values for the test dataset.

Number of epochs	100
Batch size	10
Number of perceptron	39
Optimizer	SGD
Activation function for hidden layer	elu
Activation function for output layer	Softmax
loss function	categorical cross_entropy

d Confusion Matrix

The test dataset (100K), was tested based on the above parameters (in the table) given, and the confusion matrix along with the minimum probability of error is reported below. Normalized Confusion matrix:

$$\begin{bmatrix} 0.93 & 0 & 0 & 0.07 \\ 0 & 0.77 & 0.21 & 0.02 \\ 0 & 0.12 & 0.88 & 0 \\ 0.07 & 0 & 0 & 0.93 \end{bmatrix}$$

Minimum probability of error: 0.1233
Accuracy: 87.67%
Categorical loss:0.2860

VI Dataset 5000

a Actual Data Distribution

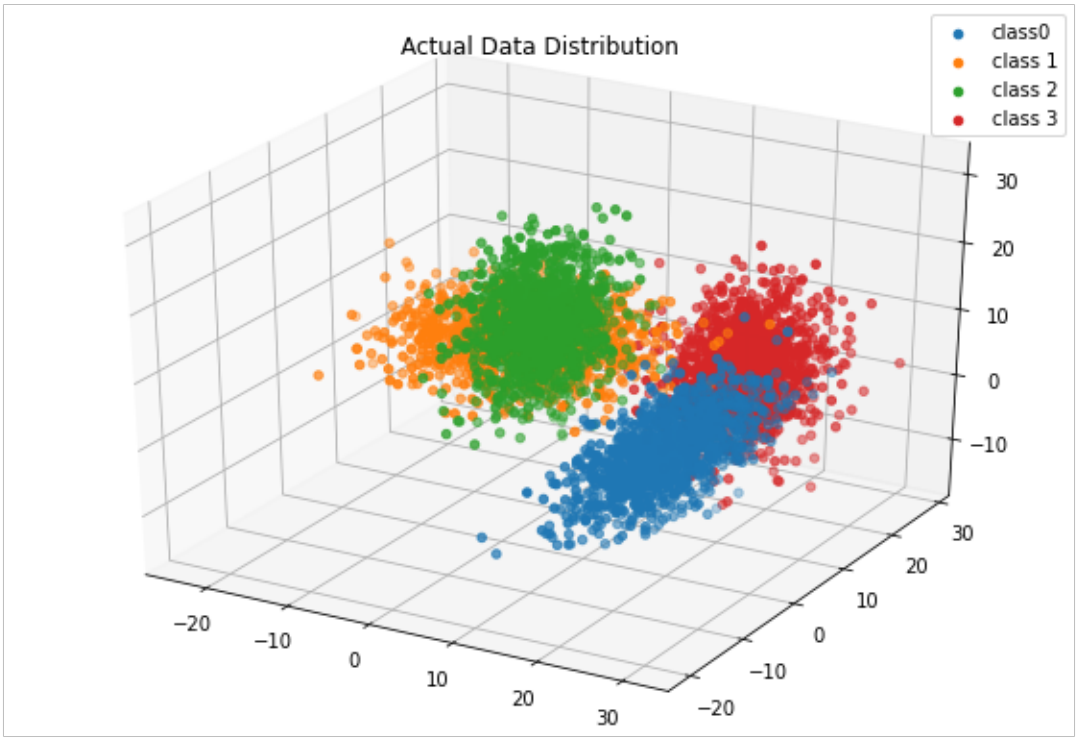

Figure 21: Actual data distribution

b Number of epochs

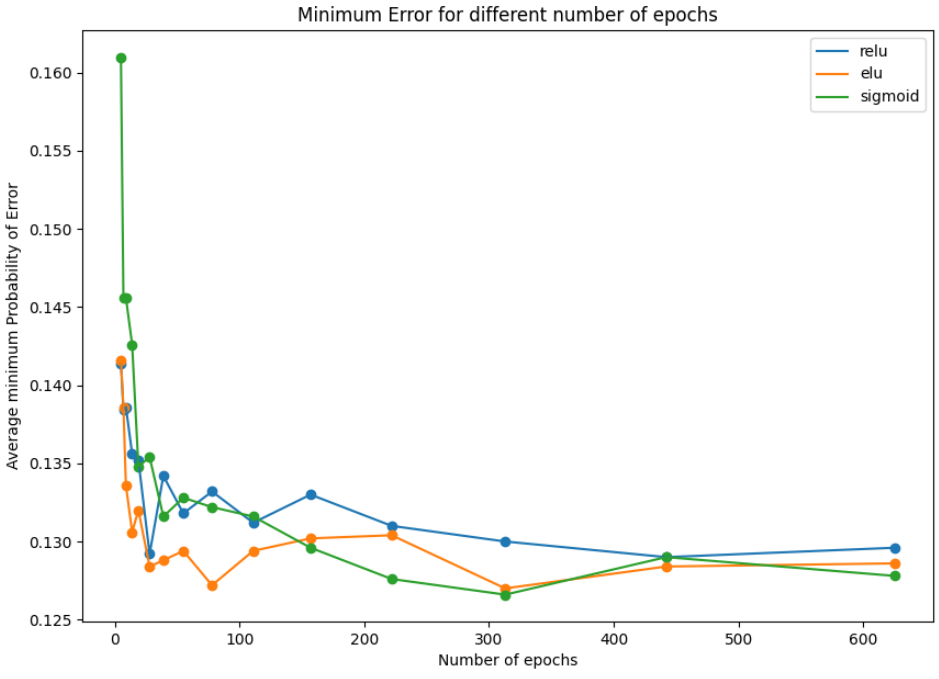

Figure 22: Minimum Error for different number of epochs

c Number of Perceptrons

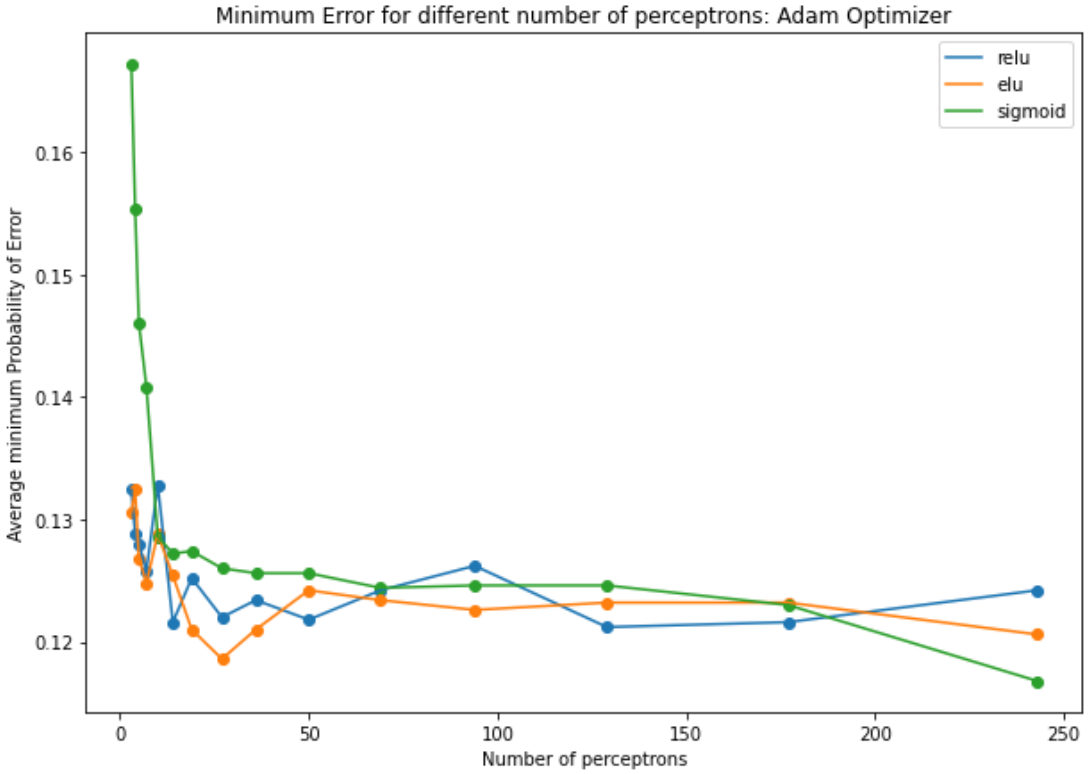

Figure 23: Minimum Error for different number of epochs under Adam optimizer

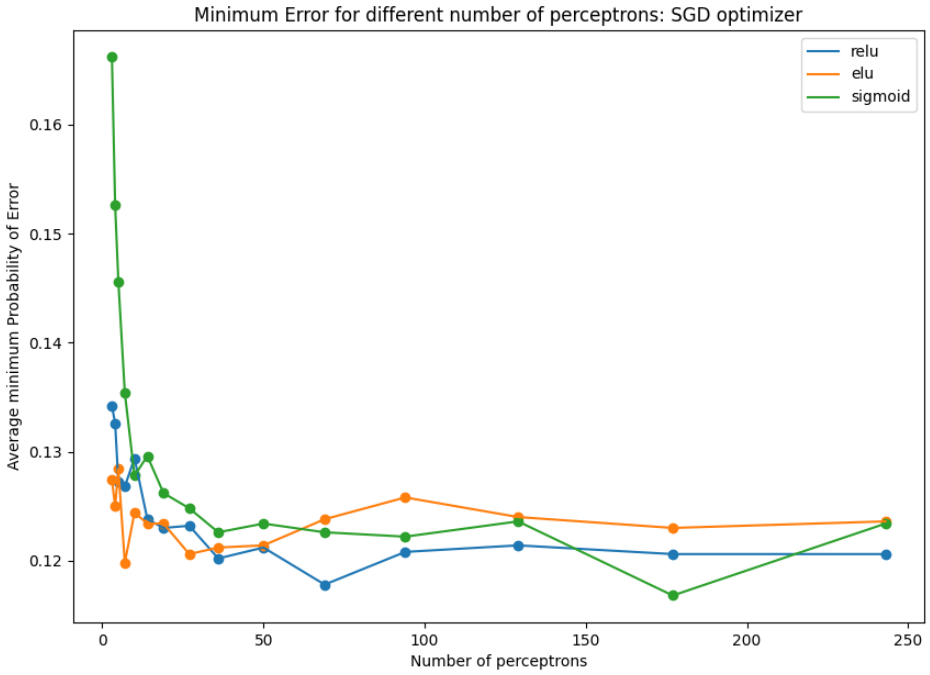

Figure 24: Minimum Error for different number of epochs under SGD optimizer

Minimum probability of error for Adam:0.1168
Minimum probability of error for SGD:0.1189

Based on the above plots and probability error, the following table is prepared and shows the selected parameter values for the test dataset.

Number of epochs	100
Batch size	10
Number of perceptron	27
Optimizer	Adam
Activation function for hidden layer	elu
Activation function for output layer	Softmax
loss function	categorical cross_entropy

d Confusion Matrix

The test dataset (100K), was tested based on the above parameters (in the table) given, and the confusion matrix along with the minimum probability of error is reported below. Normalized Confusion matrix:

$$\begin{bmatrix} 0.9 & 0 & 0 & 0.1 \\ 0 & 0.8 & 0.19 & 0.01 \\ 0 & 0.12 & 0.88 & 0 \\ 0.04 & 0 & 0 & 0.95 \end{bmatrix}$$

Minimum probability of error: 0.1185
Accuracy: 88.15%
Categorical loss:0.2714

VII Conclusion

The following plot shows the plot of the minimum probability of error for each dataset along with the theoretical optimal classifier.

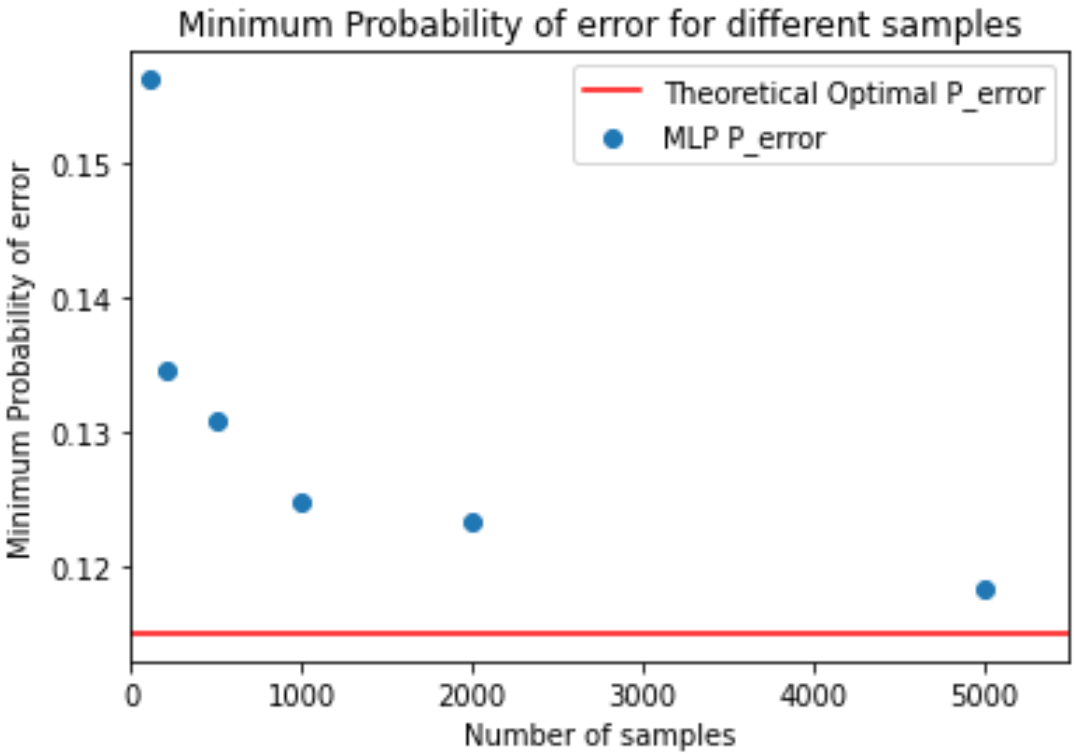

Figure 25: Minimum Error for different number of samples

Observation:

From the above figure, we can see that the optimal classifier has less probability of error than the MLP classifier in all the datasets. This is expected because in the optimal classifier, we are using the true pdf of the test dataset. Also, as the number of samples increases, the classifier is performing better this is because as the samples increases we will capture the true distribution of data and this will minimize both bias and variance.

2. **GMM**:

Given the following data:

$$priors = [0.1, 0.2, 0.3, 0.4]$$
$$number\ of\ samples = [10, 100, 1000, 10000]$$

$$m_0 = \begin{bmatrix} 0 \\ 50 \end{bmatrix} C_0 = \begin{bmatrix} 10 & 0 \\ 10 & 20 \end{bmatrix} m_1 = \begin{bmatrix} 30 \\ 50 \end{bmatrix} C_1 = \begin{bmatrix} 20 & 30 \\ 0 & 10 \end{bmatrix}$$

$$m_2 = \begin{bmatrix} 10 \\ 10 \end{bmatrix} c_2 = \begin{bmatrix} 20 & 30 \\ 0 & 10 \end{bmatrix} m_3 = \begin{bmatrix} 30 \\ 0 \end{bmatrix} C_3 = \begin{bmatrix} 10 & 20 \\ 0 & 40 \end{bmatrix}$$

Implementation:

After generating the i.i.d samples using the above covariance and mean vectors, I used the built in function *sklearn.mixture.GaussianMixture* to estimate the parameters for each Gaussian component. Expectation maximization (EM) algorithm was used in-order to maximize the expected value of the log likelihood function. To find the best Gaussian component, 10 fold cross validation was used. The performance was measured by a built-in function *sklearn.model_selection.cross_val_score*, which return log likelihood score of data. This score was used to find the best model order (which has maximum score). The above procedure was repeated for 30 times and the rate of the model orders selected in each experiment was calculated.

I Dataset 10

a Actual Data Distribution

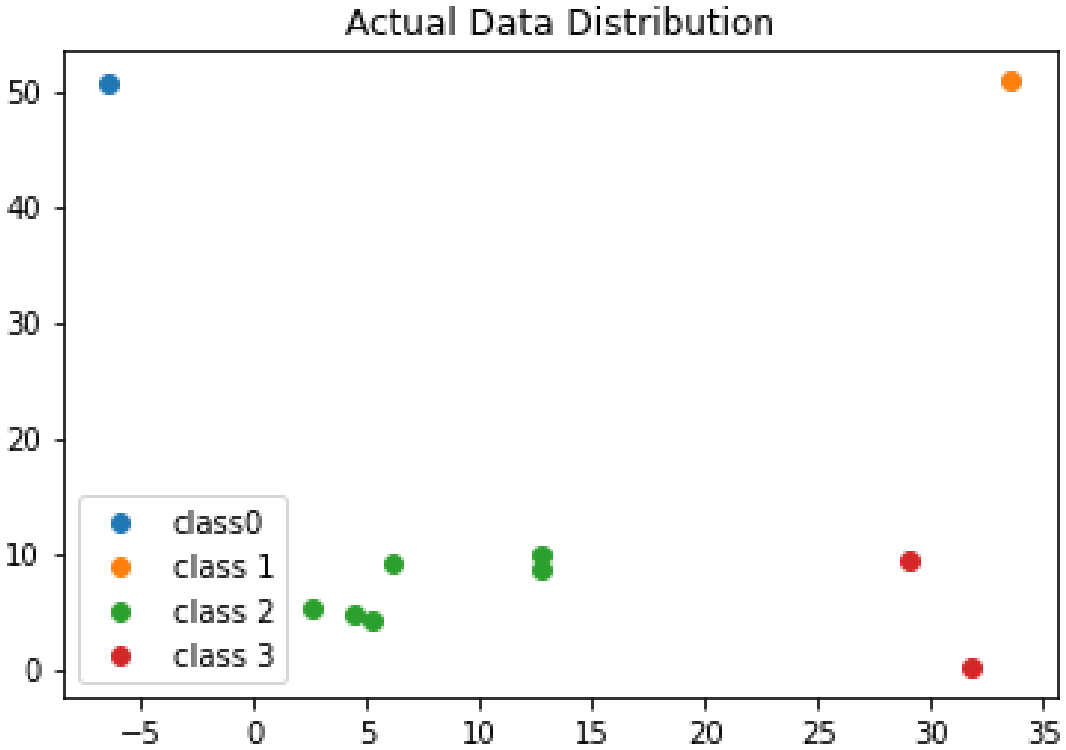

Figure 26: Actual data distribution

b Model order selection

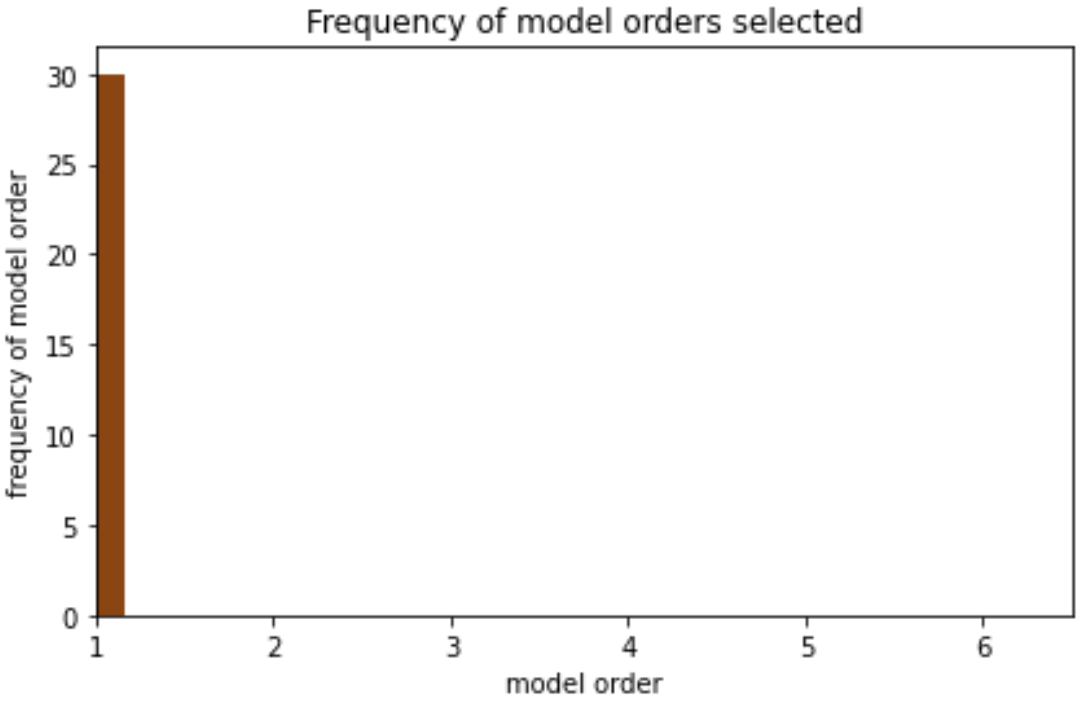

Figure 27: Actual data distribution

c Validation performance measures

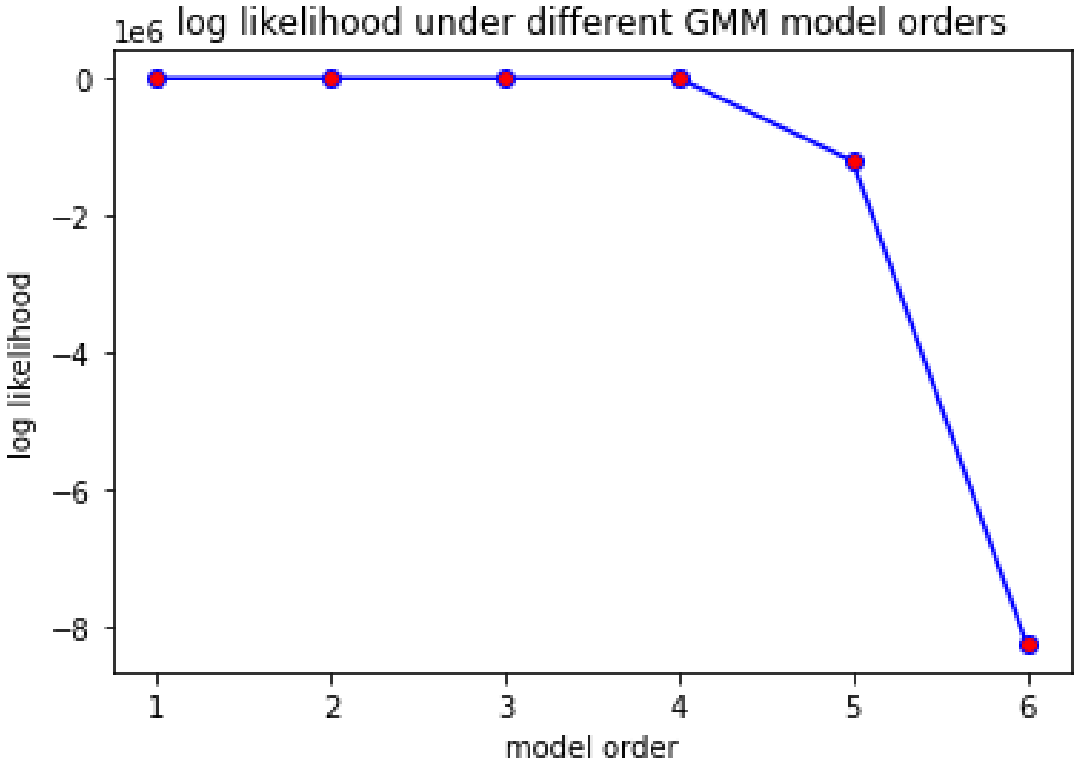

Figure 28: Actual data distribution

II Dataset 100

a Actual Data Distribution

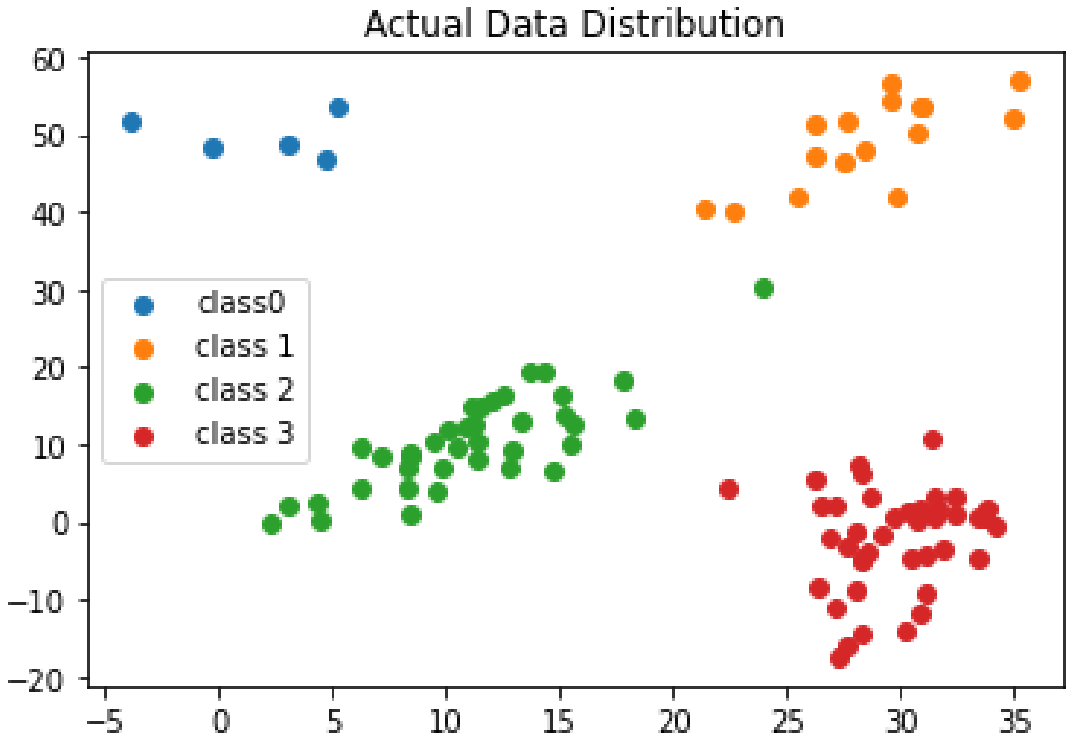

Figure 29: Actual data distribution

b Model order selection

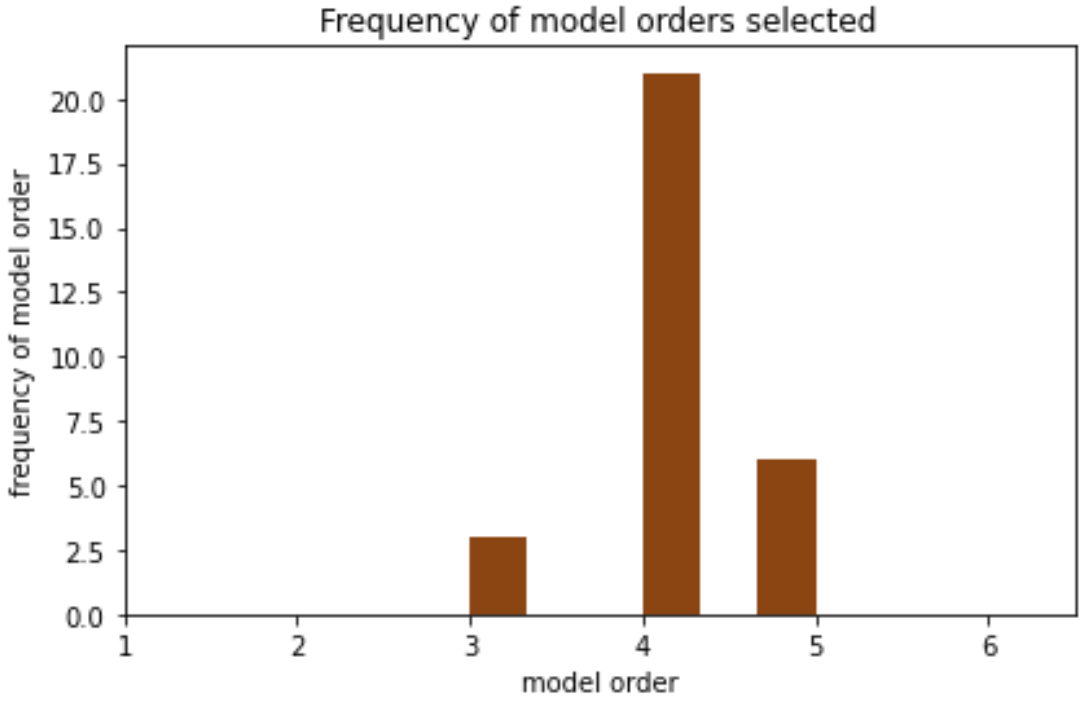

Figure 30: Actual data distribution

c Validation performance measures

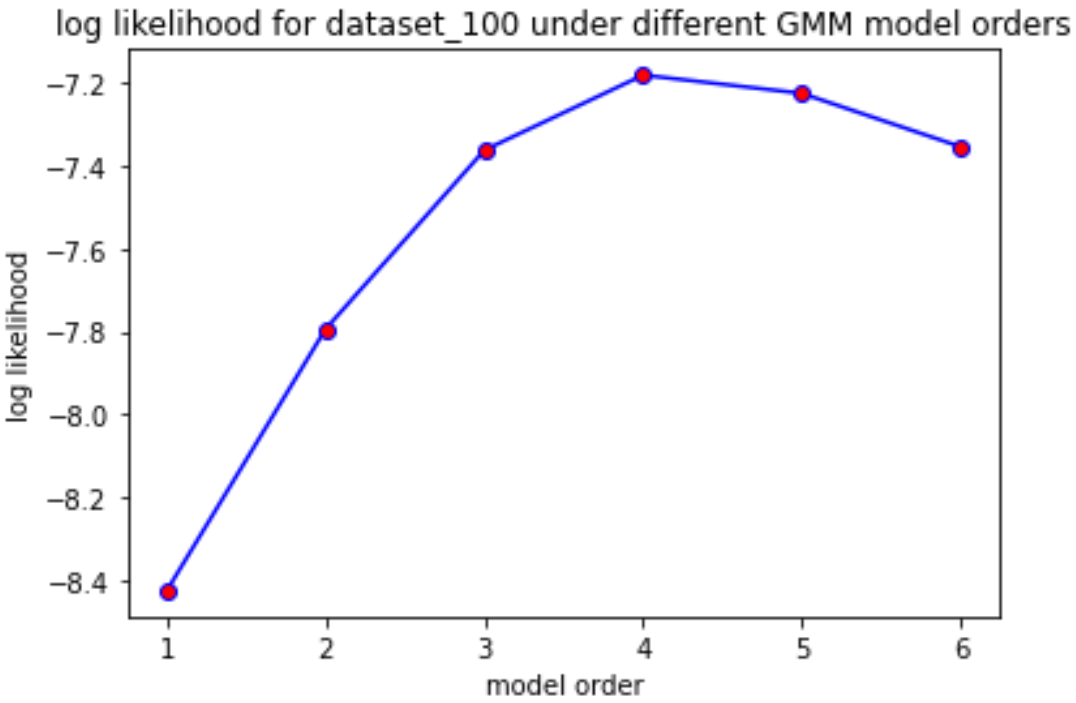

Figure 31: Actual data distribution

III Dataset 1000

a Actual Data Distribution

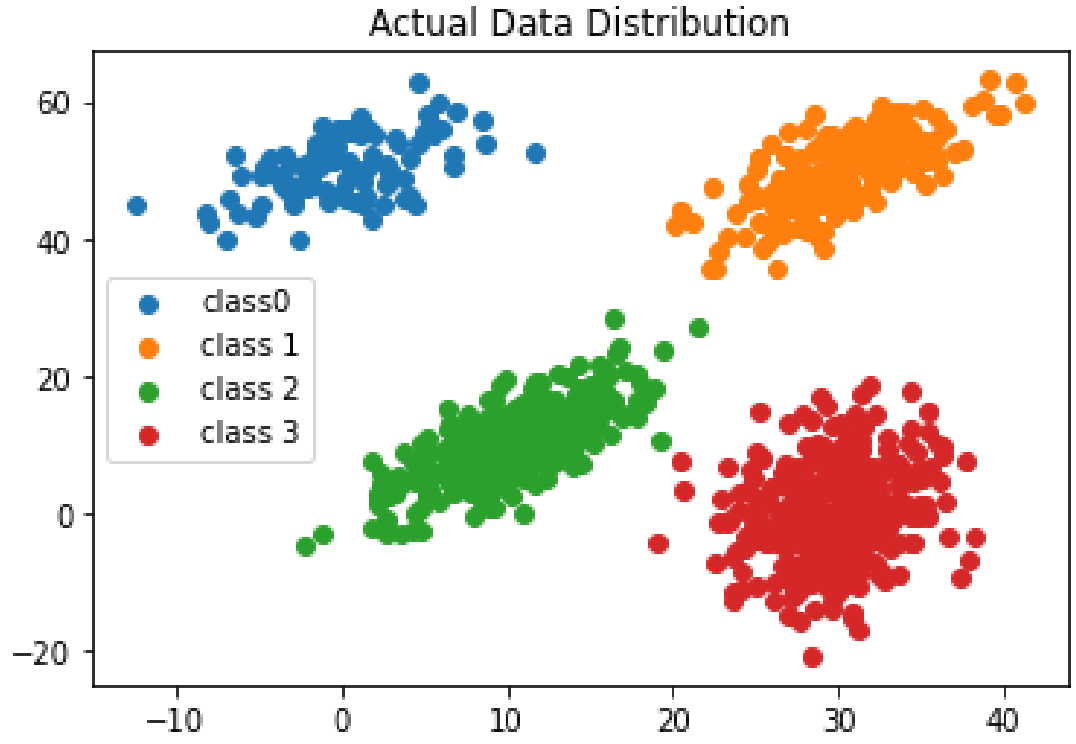

Figure 32: Actual data distribution

b Model order selection

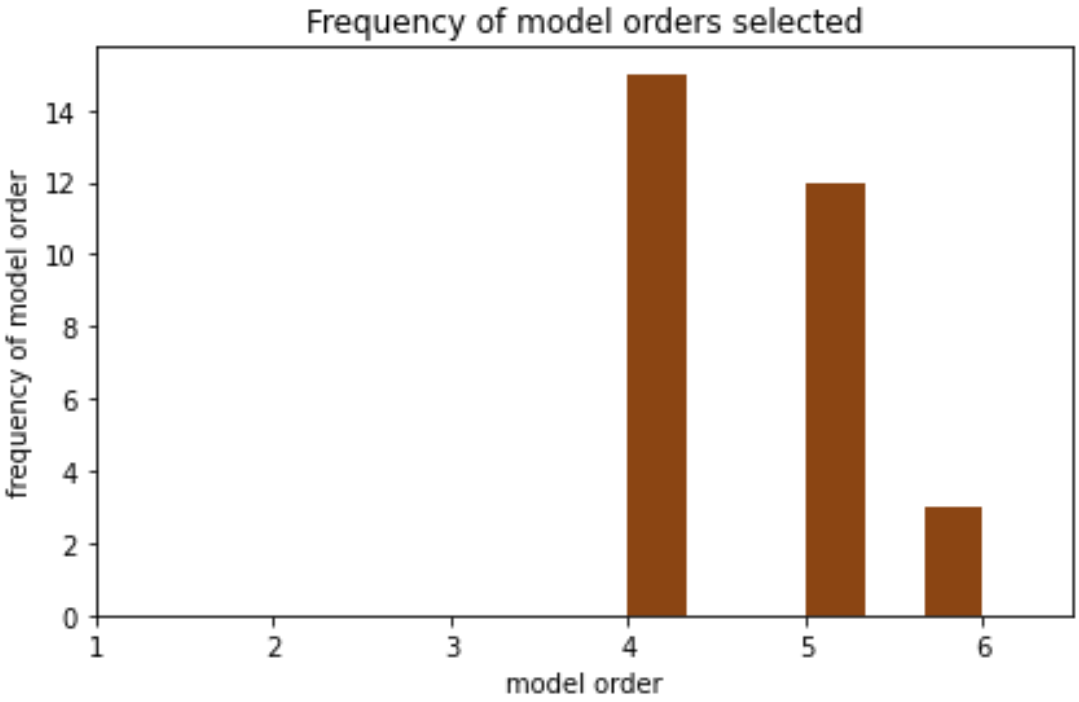

Figure 33: Actual data distribution

c Validation performance measures

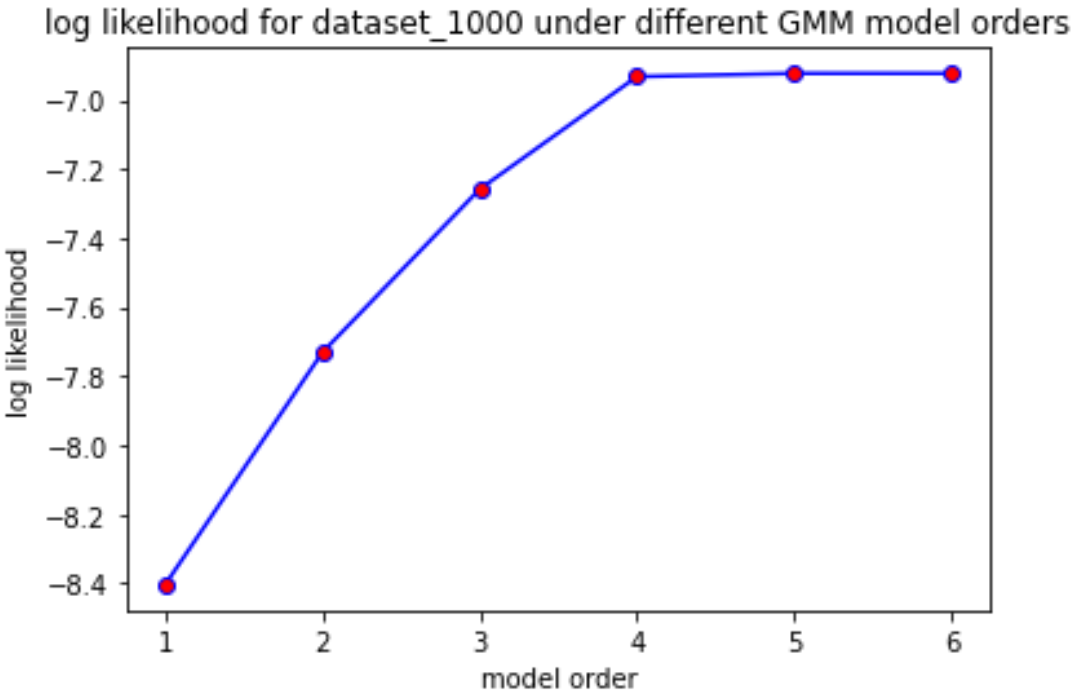

Figure 34: Actual data distribution

IV Dataset 10000

a Actual Data Distribution

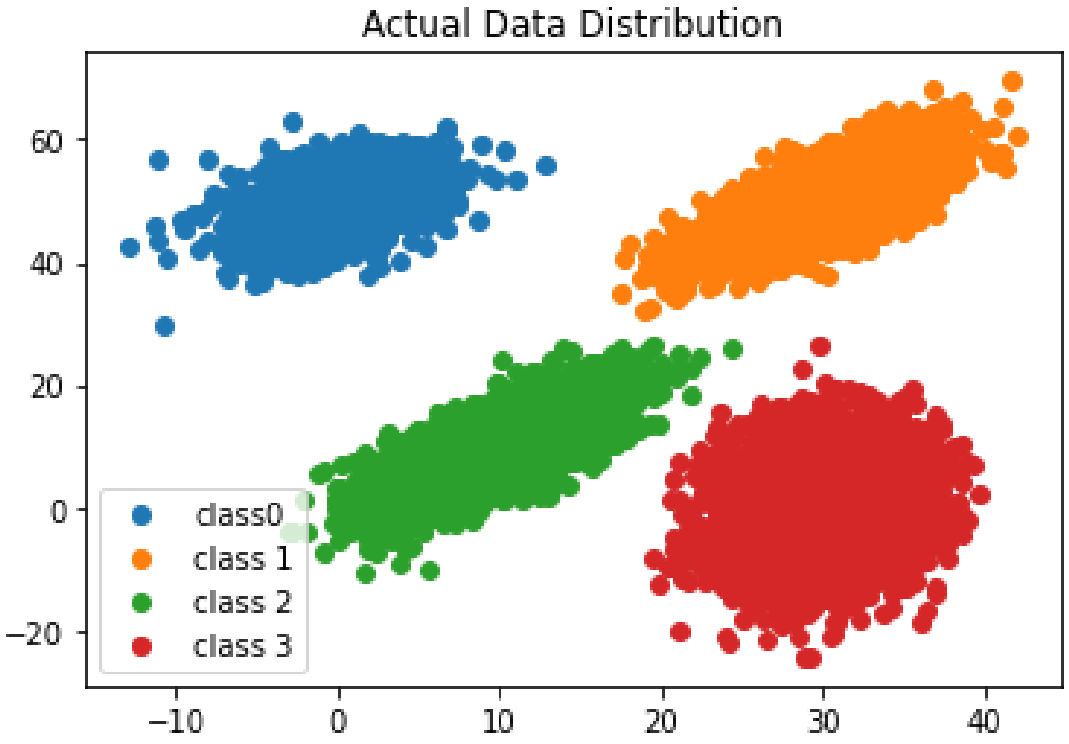

Figure 35: Actual data distribution

b Model order selection

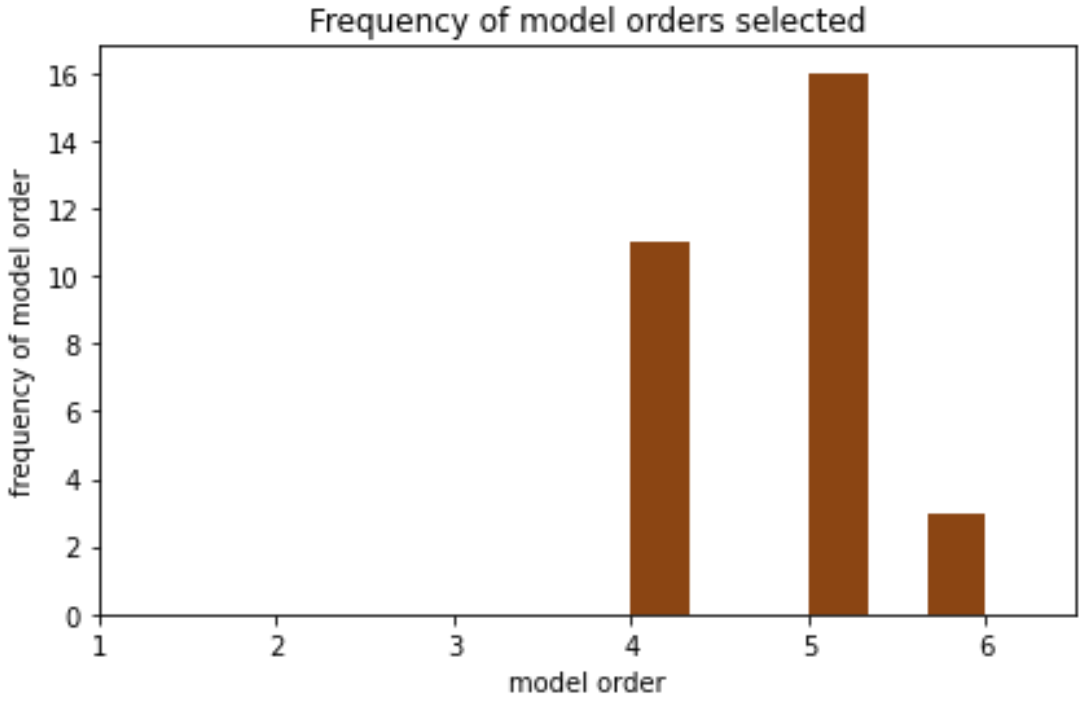

Figure 36: Actual data distribution

c Validation performance measures

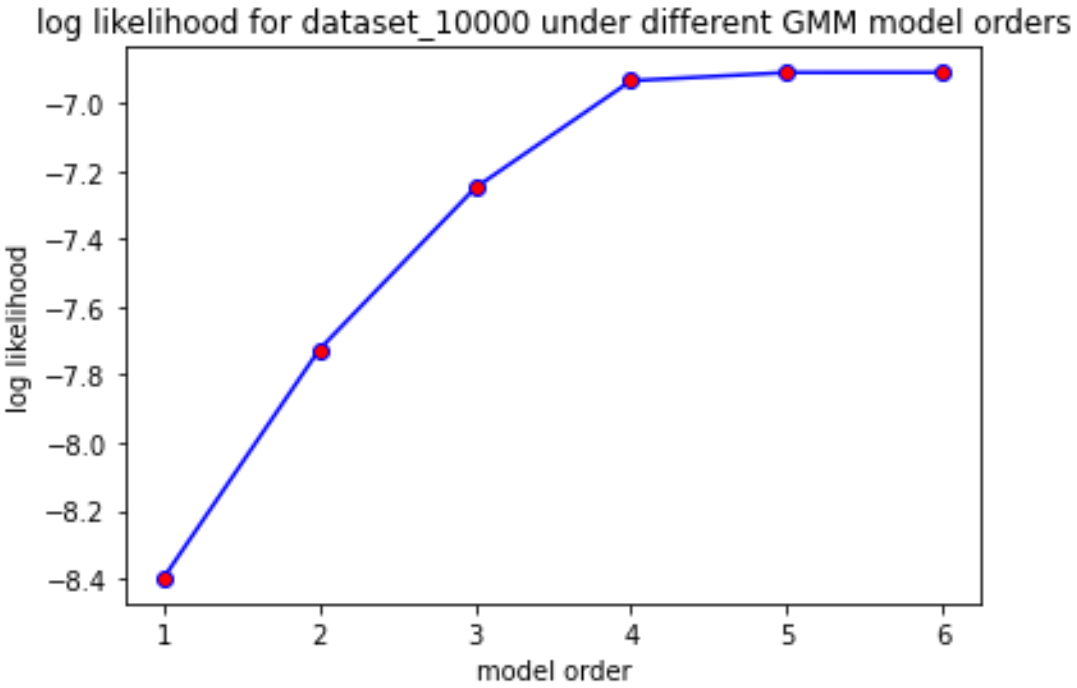

Figure 37: Actual data distribution

V Conclusion

The following table shows the rate ($\frac{frequency\ of\ model\ order}{number\ of\ experiment}$) for which the model order has been selected for 30 experiments.

Rates for the 6 model orders						
Number of Samples	1	2	3	4	5	6
10	1	0	0	0	0	0
100	0	0	0.1	0.7	0.2	0
1000	0	0	0	0.5	0.4	0.1
10000	0	0	0	0.37	0.53	0.1

Observation:

From the frequency of model order graph, we can see that for the smallest number of dataset (10 in this case), the Gaussian component selected is 1, this is expected because with 10 samples we can not fit higher Gaussian components.As the number of samples increases, the higher Gaussian components (model order) gets selected. Specifically for the 1000 and 10000, Gaussian component of 4 and 5 were selected frequently. This is because of the overlapping of the data distributions as the sample number increases and more components will be selected to fit the distribution. I have tested the algorithm with far apart data distributions (by varying their mean), and for this well separated distributions, I usually get 4 Gaussian components.
From the log likelihood graph, we can see that for small number of samples (10) the likelihood is higher for small number of Gaussian components while for those with higher samples, we can see that their log likelihood is higher

for higher Gaussian components and the curve is saturating for model orders $(4, 5, 6)$ as we increase the number of samples. If I could have used the negative log likelihood curve instead, I would have got the opposite curve but it won't have any difference while evaluating the performance measure of each model orders. Finally, the rate of the model orders selected is prepared in the above table and this is a direct numeric representation of the frequency graph.

A Appendix

```
1
2  # Question 1
3  import numpy as np
4  from scipy . stats import multivariate_normal as mvn
5  import tensorflow as tf
6  from sklearn.metrics import confusion_matrix
7  import random
8  from tensorflow.keras.utils import to_categorical
9  import matplotlib.pyplot as plt
10 from tensorflow.keras.layers import Dense
11 import matplotlib.pyplot as plt
12
13
14 def data(n):
15   label = np.zeros((1,n))
16   for i in range(n):
17     label[0,i]= np.random.choice(np.arange(4),p=prior)
18
19   x = np.zeros ((features , n))
20
21   for index in range (n) :
22     if label [0,index] == 0:
23       x [:,index] = mvn(m0.reshape(3,),C0).rvs(1)
24     elif label[0,index]==1:
25       x [:,index] = mvn(m1.reshape(3,),C1).rvs(1)
26     elif label[0,index]==2:
27       x [:,index] = mvn(m2.reshape(3,),C2).rvs(1)
28     else:
29       x [:,index] = mvn(m3.reshape(3,),C3).rvs(1)
30
31   return x,label # return Data_concat
32
33 # theoretical optimal classifier
34 def theoretical_classifier(sample_type):
35   data_opt,label_opt=data(sample_type) # sample_type=samples[6] for the
       test dataset
36   lik=np.zeros((num_class,data_opt.shape[1]))
37   tot_lik=np.zeros((num_class,data_opt.shape[1]))
38   loss_mat=np.array([[0,1,1,1],[1,0,1,1],[1,1,0,1],[1,1,1,0]])
39   for label in range(num_class):
40     samp=mvn.pdf(data_opt.T,mean=mu_vector[label],cov=sigma_vector[label
       ])
41     lik[label]=samp
42   prod=np.matmul(prior.reshape(1,num_class),lik)
43
44   for label in range(num_class):
45     tot_lik[label]=prod
46
47   temp = prior.reshape(-1, 1)*lik/tot_lik
48   risk_mat = np.matmul(loss_mat, temp)
49
50   label_sel=np.argmin(risk_mat,axis=0)
51
52   correct_samp=[]
53   incorrect_samp=[]
54
```

```
  for label in range(num_class):
    lbl=(label_opt==label)
    corr_lbl=((lbl)*(label_sel==lbl)).astype('int')
    incorr_lbl=((lbl)*(label_sel!=lbl)).astype('int')
    correct_=np.where(corr_lbl==1)[0]
    incorrect_=np.where(incorr_lbl==1)[0]

    correct_samp.append(correct_)
    incorrect_samp.append(incorrect_)

  p_err=1.0*np.sum((label_sel!=label_opt).astype('int'))/data_opt.shape
    [1]
  print(f'Theoretical Minimum Probability of Error:{p_err.round(4)}')

def split_data(data, folds):
  data_split = []
  random.seed(1)
  data_temp = list(data)
  fold_size = int(len(data) / folds)
  for i in range(folds):
    fold = list()
    while len(fold) < fold_size:
      rand_index = random.randrange(len(data_temp))
      fold.append(data_temp.pop(rand_index))
    data_split.append(fold)
  return data_split

def test_train_split(folds,i):
  test=np.array(folds.pop(i))
  train=np.array(folds)
  xtest=test[:,0:3]
  ytest=test[:,3]
  xtrain=train[:,:,0:3].reshape(-1,3)
  ytrain=train[:,:,3].reshape(-1)

  return xtrain,ytrain,xtest,ytest

def perceptrons_sel(sample_type):

  error_mat=np.zeros((len(percept_list),len(activations)))
  accuracy_mat=np.zeros((len(percept_list),len(activations)))

  weights_perceptron_layer1=[]
  weights_perceptron_layer2=[]
  data_train,label_train=data(sample_type)
  data_concat=np.hstack((data_train.T,label_train.T))

  for activ in range(len(activations)):

    activ_weight1=[]
    activ_weight2=[]

    for idx,percpt in enumerate(percept_list):
      err_fold=[]
      acc_fold=[]

      for i in range(num_folds):
        X_train,Y_train,X_test,Y_test=test_train_split(split_data(
```

```
    data_concat,num_folds),i)
112         input_shape = (features,)
113         Y_train_encod=to_categorical(Y_train,num_class)
114         model = tf.keras.models.Sequential()
115 
116         model.add(Dense(percpt,kernel_initializer='random_uniform',
    input_shape=input_shape, activation=activations[activ]))
117         model.add(Dense(num_class, kernel_initializer='random_uniform',
    activation='softmax'))
118 
119         model.compile( optimizer='Adam',loss='categorical_crossentropy',
     metrics=['accuracy'])
120         model.fit(X_train,Y_train_encod, epochs=10, batch_size=10,
    verbose=0)
121         temp_y=model.predict(X_test)
122         y_pred=np.argmax(temp_y, axis=1)
123         tmt=to_categorical(Y_test,num_class)
124         y_test=np.argmax(tmt, axis=1)
125 
126         cm = confusion_matrix(y_test, y_pred,labels=[0.0,1.0,2.0,3.0])
127         temp=(cm[0,0]+cm[1,1]+cm[2,2]+cm[3,3])/cm.sum()
128         err_fold.append(1-temp)
129         acc_fold.append(temp)
130 
131       error_mat[idx][activ]=(np.mean(err_fold)) # choose the perceptron
    based on the min of error
132       accuracy_mat[idx][activ]=(np.mean(acc_fold))
133 
134       activ_weight1.append((model.layers[0].get_weights())[0])
135       activ_weight2.append((model.layers[1].get_weights())[0])
136     weights_perceptron_layer1.append(activ_weight1)
137     weights_perceptron_layer2.append(activ_weight2)
138 
139 # test the new data
140  # find the perceptron with minimum error or loss
141 opt_percerptron=np.where(error_mat==error_mat.flat[np.argmin(error_mat
    )])
142 optimal_perceptron=percept_list[int(opt_percerptron[0])]
143 optimal_activation=activations[int(opt_percerptron[1])]
144 
145 print(f'activation:{optimal_activation}')
146 print(f'optimal number of neurons:{optimal_perceptron}')
147 
148 init1=  tf.constant_initializer(weights_perceptron_layer1[int(
    opt_percerptron[1])][int(opt_percerptron[0])])
149 init2=  tf.constant_initializer(weights_perceptron_layer2[int(
    opt_percerptron[1])][int(opt_percerptron[0])])
150 
151 
152 data_test,label_test=data(samples[6]) # test dataset
153 X_train_=data_train.T
154 X_test_=data_test.T
155 
156 Y_train_ = to_categorical(label_train.T,num_class)
157 Y_test_ = to_categorical(label_test.T, num_class)
158 
159 X_train_ = X_train_.reshape(X_train_.shape[0], features)
160 X_test_ = X_test_.reshape(X_test_.shape[0], features)
```

```
161
162   input_shape = (features,)
163   print(f'Feature shape: {input_shape}')
164
165   model = tf.keras.Sequential()
166   model.add(Dense(optimal_perceptron,kernel_initializer=init1,
        input_shape=input_shape, activation=optimal_activation))
167   model.add(Dense(num_class, kernel_initializer=init2,activation='
        softmax'))
168   model.compile(loss='categorical_crossentropy', optimizer='adam',
        metrics=['accuracy'])
169   model.fit(X_train_, Y_train_, epochs=10, batch_size=32, verbose=1)
170
171
172   test_results = model.evaluate(X_test_, Y_test_, verbose=1)
173   print(f'Test results - Loss: {test_results[0]} - Accuracy: {
        test_results[1]}')
174
175   y_pred_=model.predict(X_test_)
176   y_pred_=np.argmax(y_pred_, axis=1)
177   y_test_=np.argmax(Y_test_, axis=1)
178   cm = confusion_matrix(y_test_, y_pred_)
179   normalized_cm=cm / cm.astype(float).sum(axis=1)
180   error = (1- (cm[0,0]+cm[1,1]+cm[2,2]+cm[3,3])/cm.sum())
181
182   print(f'minimum probability of error:{error}')
183   print(f'confusion matrix:{cm}')
184   print(f'Normalized confusion matrix:{normalized_cm}')
185
186   return data_train,label_train,error_mat
187
188 def plot_results(sample_type):
189   dataset_train,labels_training,error_matt=perceptrons_sel(sample_type)
190   #plot actual data distribution
191   actual_data=np.array(dataset_train.T)
192   actual_label=np.array(labels_training.T)
193
194   x0=[actual_data[i] for i in range(sample_type) if actual_label[i]==0]
195   x1=[actual_data[i] for i in range(sample_type) if actual_label[i]==1]
196   x2=[actual_data[i] for i in range(sample_type) if actual_label[i]==2]
197   x3=[actual_data[i] for i in range(sample_type) if actual_label[i]==3]
198
199   fig = plt.figure(figsize = (10, 7))
200   ax = plt.axes(projection ="3d")
201
202   ax.scatter3D((np.array(x0))[:,0],(np.array(x0))[:,1],(np.array(x0))
        [:,2])
203   ax.scatter3D((np.array(x1))[:,0],(np.array(x1))[:,1],(np.array(x1))
        [:,2])
204   ax.scatter3D((np.array(x2))[:,0],(np.array(x2))[:,1],(np.array(x2))
        [:,2])
205   ax.scatter3D((np.array(x3))[:,0],(np.array(x3))[:,1],(np.array(x3))
        [:,2])
206   plt.legend(['class0','class 1','class 2','class 3'])
207   plt.title('Actual Data Distribution')
208   plt.show
209
210   #plot min error vs number of perceptrons
```

```
211
212   tt=np.array(percept_list)
213   bar1=error_matt[:,0]
214   bar2=error_matt[:,1]
215   bar3=error_matt[:,2]
216
217   fig = plt.figure(figsize = (10, 7))
218   ax2 = plt.axes()
219   ax2.scatter(tt,bar1)
220   ax2.plot(tt,bar1)
221
222   ax2.scatter(tt,bar2)
223   ax2.plot(tt,bar2)
224
225   ax2.scatter(tt,bar3)
226   ax2.plot(tt,bar3)
227
228   plt.legend(activations)
229   plt.xlabel('Number of perceptrons')
230   plt.ylabel('Average minimum Probability of Error')
231   plt.title('Minimum Error for different number of neurons')
232   plt.show()
233
234
235 def main():
236   plot_results(samples[0])
237   theoretical_classifier(samples[6])
238 if __name__ == "__main__":
239   prior=np.array([0.25,0.25,0.25,0.25])
240   features=3
241   num_class=4
242   samples = [100,200,500,1000,2000,5000,100000]
243   m0 = np.array([20, 0, 0])
244   m1 = np.array([0, 5, 10])
245   m2 = np.array([5, 0, 15])
246   m3 = np.array([20, 15,5])
247
248   mu_vector = [m0, m1, m2, m3]
249
250   C0 = np.array([[10, 0, 10], [5, 40, 0], [5, 0, 20]])
251   C1 = np.array([[40, 0, 0], [0, 25, 0], [0, 0, 10]])
252   C2 = np.array([[10, 0, 0], [0, 20, 0], [0, 0, 30]])
253   C3 = np.array([[15, 0, 0], [0, 20, 10], [0, 0, 25]])
254   sigma_vector = [C0, C1, C2, C3]
255
256   activations=['relu','elu','sigmoid']
257   num_folds=10
258   percept_list=np.logspace(1,3,num = 20,endpoint = True,base = 5,dtype =
        int)
259   percept_list=percept_list.tolist()
260   main()
```

Listing 1: Question 1

B Appendix

```python
# Question 2
import numpy as np
import random
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
from sklearn.model_selection import cross_val_score
from scipy . stats import multivariate_normal as mvn


def data(n):
  label = np.zeros((1,n))

  for i in range(n):
    label[0,i]=np.random.choice(np.arange(4),p=prior)

  x = np.zeros ((features , n))
  for i in range (n) :
    if label[0,i]==0:
      x [:,i] = mvn(m0.reshape(2,),C0).rvs(1)
    elif label[0,i]==1:
      x [:,i] = mvn(m1.reshape(2,),C1).rvs(1)
    elif label[0,i]==2:
      x [:,i] = mvn(m2.reshape(2,),C2).rvs(1)
    else:
      x [:,i] = mvn(m3.reshape(2,),C3).rvs(1)

  return x,label

def model_order(sample_type):# model order selection
  pos_log=[]
  rates=[]
  for k in range(num_expt):
    Data_train,label=data(sample_type)
    gmm_avg=np.zeros((len(gmm_list)))
    gmm_var=np.zeros((len(gmm_list)))
    temp=[]
    for idx,i in enumerate(gmm_list):
      gmm=GaussianMixture(i,covariance_type='full',random_state=None)
      scor=cross_val_score(gmm,Data_train.T,cv=num_fold)
      avg_scor=np.mean(scor)
      var_scor=np.std(scor)

      gmm_avg[idx]=avg_scor
      gmm_var[idx]=var_scor

      gmm.fit(Data_train.T)
      temp.append(avg_scor)

    model_sel=np.argmax(gmm_avg) + 1
    model_sel_list[k]=int(model_sel)
    pos_log.append(temp)

# calculating the rates
  for i in range(len(gmm_list)):
    temp_list=(list(model_sel_list).count(i+1))/num_expt
    rates.append(round(temp_list,2))

  print(f'rates_{sample_type}:{rates}' )
```

```
  return model_sel_list,pos_log

def plot_results(sample_type):

  #plot the actual data distribution
  dataset_train,labels_training=data(sample_type)
  actual_data=np.array(dataset_train.T)
  actual_label=np.array(labels_training.T)

  x0=[actual_data[i] for i in range(sample_type) if actual_label[i]==0]
  x1=[actual_data[i] for i in range(sample_type) if actual_label[i]==1]
  x2=[actual_data[i] for i in range(sample_type) if actual_label[i]==2]
  x3=[actual_data[i] for i in range(sample_type) if actual_label[i]==3]

  plt.scatter((np.array(x0))[:,0],(np.array(x0))[:,1])
  plt.scatter((np.array(x1))[:,0],(np.array(x1))[:,1])
  plt.scatter((np.array(x2))[:,0],(np.array(x2))[:,1])
  plt.scatter((np.array(x3))[:,0],(np.array(x3))[:,1])
  plt.legend(['class0','class 1','class 2','class 3'])
  plt.title('Actual Data Distribution')
  plt.show

#second plot
  selected_list,pos_list=model_order(sample_type)
  n_bins=len(gmm_list)
  fig,ax=plt.subplots(tight_layout=True)
  ax.set_xlim([1,6.5])
  ax.hist(selected_list,bins=n_bins,color='saddlebrown')
  plt.xlabel('model order')
  plt.ylabel('frequency of model order')
  plt.title('Frequency of model orders selected')
  plt.show()

#third plot
  vv=np.array(pos_list) # return the pos_log
  loss_=[np.mean(vv[:,i]) for i in range(vv.shape[1]) ]
  plt.plot(np.arange(1,len(gmm_list)+1),loss_,c='b', mfc='red',marker='o
    ')
  plt.xlabel('model order')
  plt.ylabel(f'log likelihood')
  plt.title(f'log likelihood for dataset_{sample_type} under different
    GMM model orders',fontsize=12.)
  plt.show()

def main():
  plot_results(samples[0])
  plot_results(samples[1])
  plot_results(samples[2])
  plot_results(samples[3])


if __name__ == "__main__":

    prior=np.array([0.1,0.2,0.3,0.4])
    features=2
    num_class=4
    samples = [10,100,1000,10000]
    m0 = np.array([0,50])
```

```
    m1 = np.array([30,50])
    m2 = np.array([0,0])
    m3 = np.array([30, 0])

    mu_vector = [m0, m1, m2, m3]

    C0 = np.array([[10, 0], [10, 20]])
    C1 = np.array([[20, 30], [0, 10]])
    C2 = np.array([[20, 30], [0, 10]])
    C3 = np.array([[10, 20], [0, 40]])

    sigma_vector = [C0, C1, C2, C3]

    gmm_list=[1,2,3,4,5,6]
    num_expt=30
    num_fold=10
    model_sel_list=np.zeros((num_expt))
    main()
```

Listing 2: Question 2