The Gaussian class conditional pdf for 3D real valued random vector $X$ is

$$P(X) = P(L=0)\,P(X|L=0) + P(L=1)\,P(X|L=1) + P(L=2)\,P(X|L=2) + P(L=3)\,P(X|L=3)$$

Since the class priors are uniform,

$$P(L=0) = P(L=1) = P(L=2) = P(L=3) = 0.25$$

for class label 0:

$$P(L=0) = 0.25$$

$$P(X|L=0) = g(X|m_0, C_0)$$

$$m_0 = \begin{bmatrix} 2.2 \\ 0 \\ 0 \end{bmatrix} \qquad C_0 = \begin{bmatrix} 1 & -0.5 & 0.3 \\ -0.5 & 1 & -0.5 \\ 0.3 & -0.5 & 1 \end{bmatrix}$$

for class label 1:

$$P(L=1) = 0.25$$

$$P(X|L=1) = g(X|m_1, C_1)$$

$$m_1 = \begin{bmatrix} 2.2 \\ 2.2 \\ 0 \end{bmatrix} \qquad C_1 = \begin{bmatrix} 1 & 0.5 & 0 \\ 0.5 & 1 & 0.5 \\ 0 & 0.5 & 1 \end{bmatrix}$$

for class label 2:

$$P(L=2) = 0.25$$

$$P(X|L=2) = g(X|m_2, C_2)$$

$$m_2 = \begin{bmatrix} 2.2 \\ 0 \\ 2.2 \end{bmatrix} \qquad C_2 = \begin{bmatrix} 1 & 0.3 & -0.2 \\ 0.3 & 1 & 0.3 \\ -0.2 & 0.3 & 1 \end{bmatrix}$$

For class label 3:

$$P(L=3) = 0.25$$

$$P(X|L=3) = g(X|m_3, C_3)$$

$$m_3 = \begin{bmatrix} 0 \\ 2.2 \\ 0 \end{bmatrix} \qquad C_3 = \begin{bmatrix} 1 & 0.3 & 0.6 \\ 0.3 & 1 & 0.3 \\ 0.6 & 0.3 & 1 \end{bmatrix}$$

Hence, we have generated a 3-dimensional real value random vector X from 4 different classes having uniform class priors.

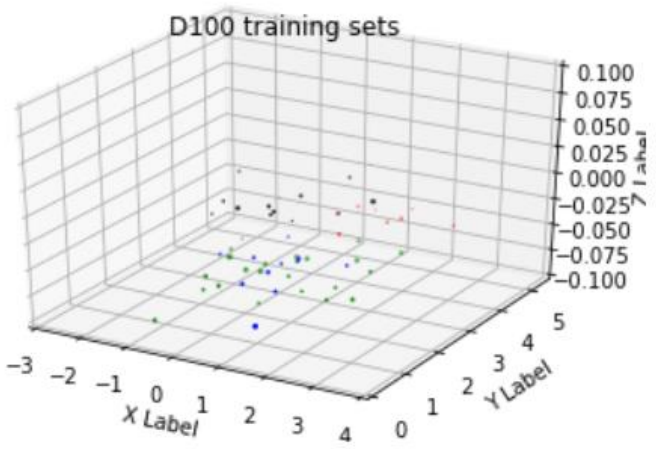
While choosing the parameters (mean & covariance), the MAP classifier that uses the true data pdf obtained the following probability of error with the given number of input samples:

| Number of Samples | Probability of Error |
|---|---|
| 100 | 0.17 |
| 200 | 0.15 |
| 500 | 0.18 |
| 1000 | 0.186 |
| 2000 | 0.1745 |
| 5000 | 0.1732 |
| 100000 (Test dataset) | 0.17567 |

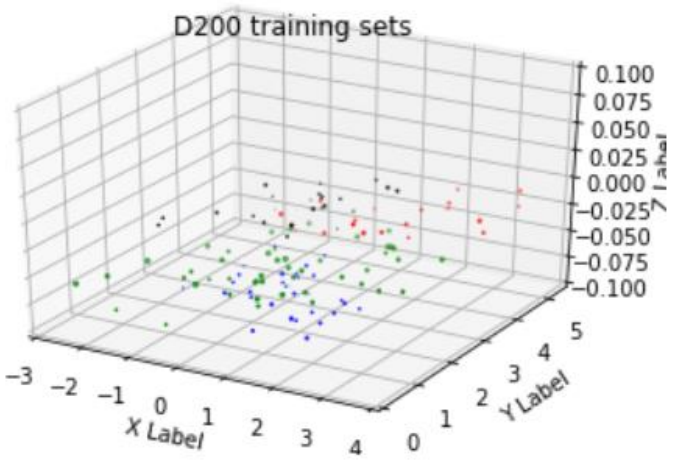
Therefore, from the table above, we can say that the probability of error lies within the 10-20 % range for the different number of samples for the set means and covariance matrices.

For the training datasets, having 100, 200, 500, 1000, 2000 and 5000 number of samples, 3-dimensional scatter plots have been generated. A scatter plot for the 100k-test dataset has also been generated and shown below.

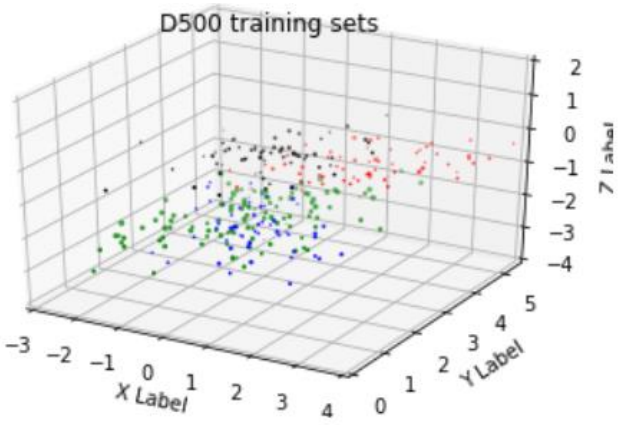
1. D100 Training Datasets (Class 0 : Blue, Class 1 : Red, Class 2: Green, Class 3 : Black)



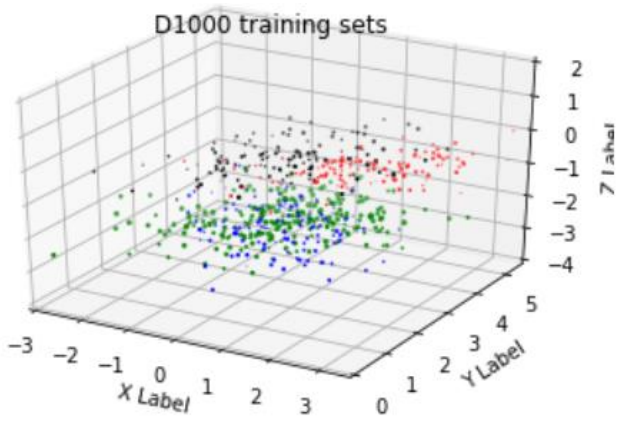

2. D200 Training Datasets (Class 0 : Blue, Class 1 : Red, Class 2: Green, Class 3 : Black)



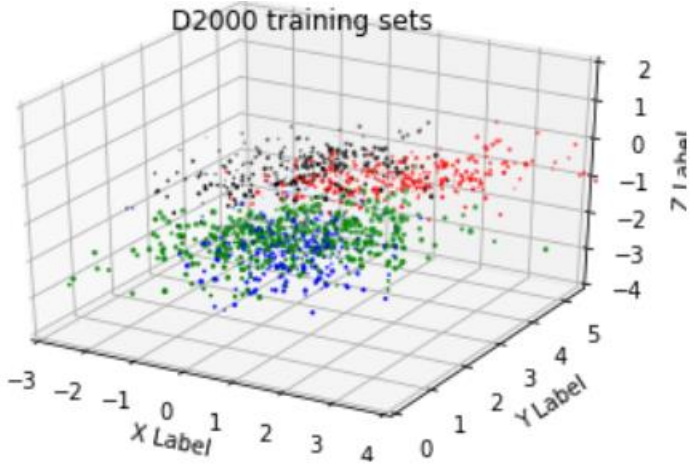
3. D500 Training Datasets (Class 0 : Blue, Class 1 : Red, Class 2: Green, Class 3 : Black)



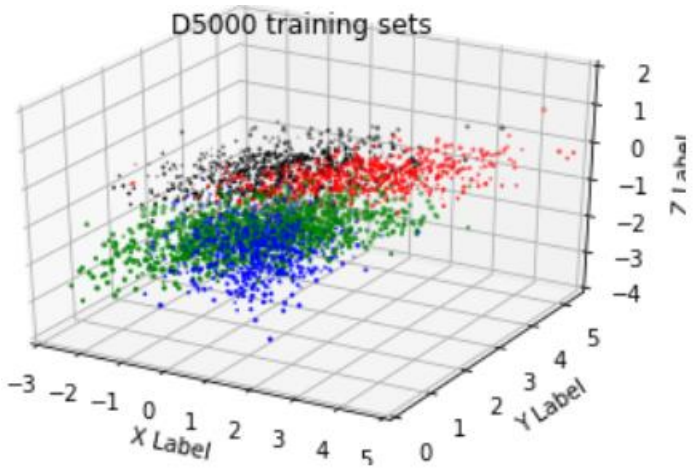

4. D1000 Training Datasets (Class 0 : Blue, Class 1 : Red, Class 2: Green, Class 3 : Black)



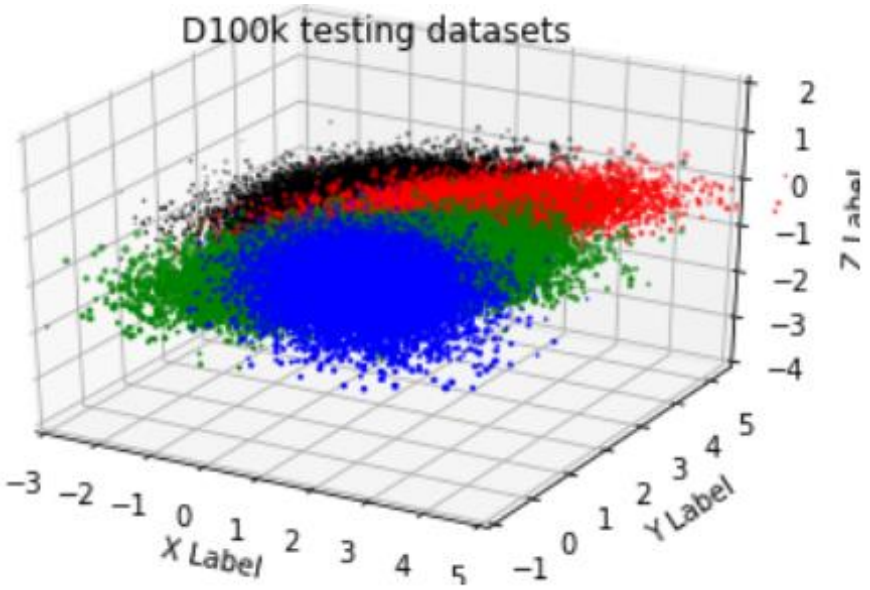

5. D2000 Training Datasets (Class 0 : Blue, Class 1 : Red, Class 2: Green, Class 3 : Black)



6. D5000 Training Datasets (Class 0 : Blue, Class 1 : Red, Class 2: Green, Class 3 : Black)




7. Test Dataset 100k (Class 0 : Blue, Class 1 : Red, Class 2: Green, Class 3 : Black)




After generating the datasets and plotting the 3-D scatter plot, we must obtain the optimum perceptron number needed for the best performance (least possible error obtained). This is done by the help of K fold (in this case, k=10) cross validation. The cross-validation loop also trains the model with the MLP, and we obtain the errors obtained with their respective number of perceptron's. Once this optimum number of perceptron's are finalized, the final model is trained with

the optimal number and the  training dataset. At the end, the trained model performance analysis is obtained by applying it on the test dataset (100k samples). The probability of error is also obtained and calculated to evaluate the performance of the model overall.

While calculating the optimum number of perceptron's and error obtained, the following software packages were implemented:

- K fold: This sk learn software package is used to form K folds and split the dataset into the training index and validation index. With the help of this, we divided the input (ie the matrix formed by generating random samples of the given number of training datasets) into training and testing.

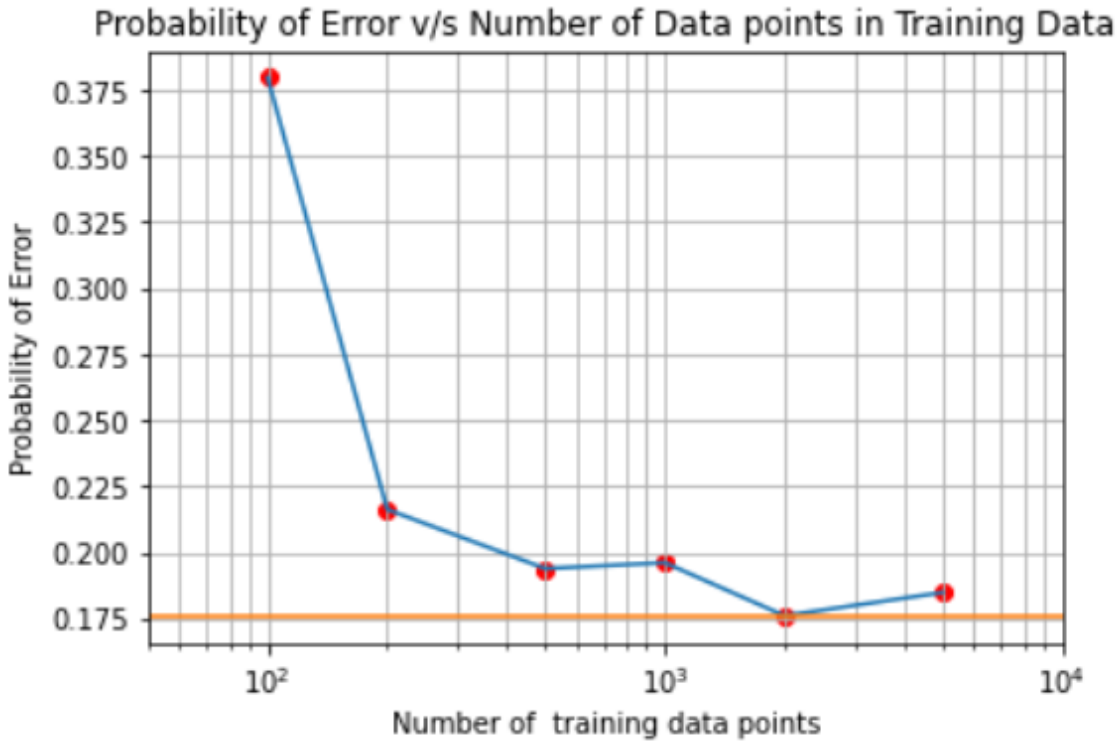
- MLPClassifier: This stands for Multi-Layer Perceptron Classifier, and was used inside the loop which was formed for obtaining K Fold cross validation of the training dataset of different number of samples (100, 200, 500, 1000, 2000 & 5000). There are a few values of p (ideal number of perceptron's needed) given in the form of a list from where it checks which 'p' gives the least value.
  There are 2 hidden layers given (p,4), one for the number of perceptron and the other for the number of classes (which is 4 in this case). As needed, the activation function is set to 'ReLU', and the data to be trained is added to the MLP classifier using .fit on the classifier. Next, the out activation is set as 'SoftMax', as mentioned in the question.  The score method is used to determine the correct and incorrect values obtained after training the neural network for the class posteriors. In this manner, by using the software packages and giving input for the activation function, number of hidden layers, output function & score calculation, we can say that it is performing the task we wanted it to.

  Here is a tabular form of the optimum number of perceptron's required, along with the size of input samples and error obtained:

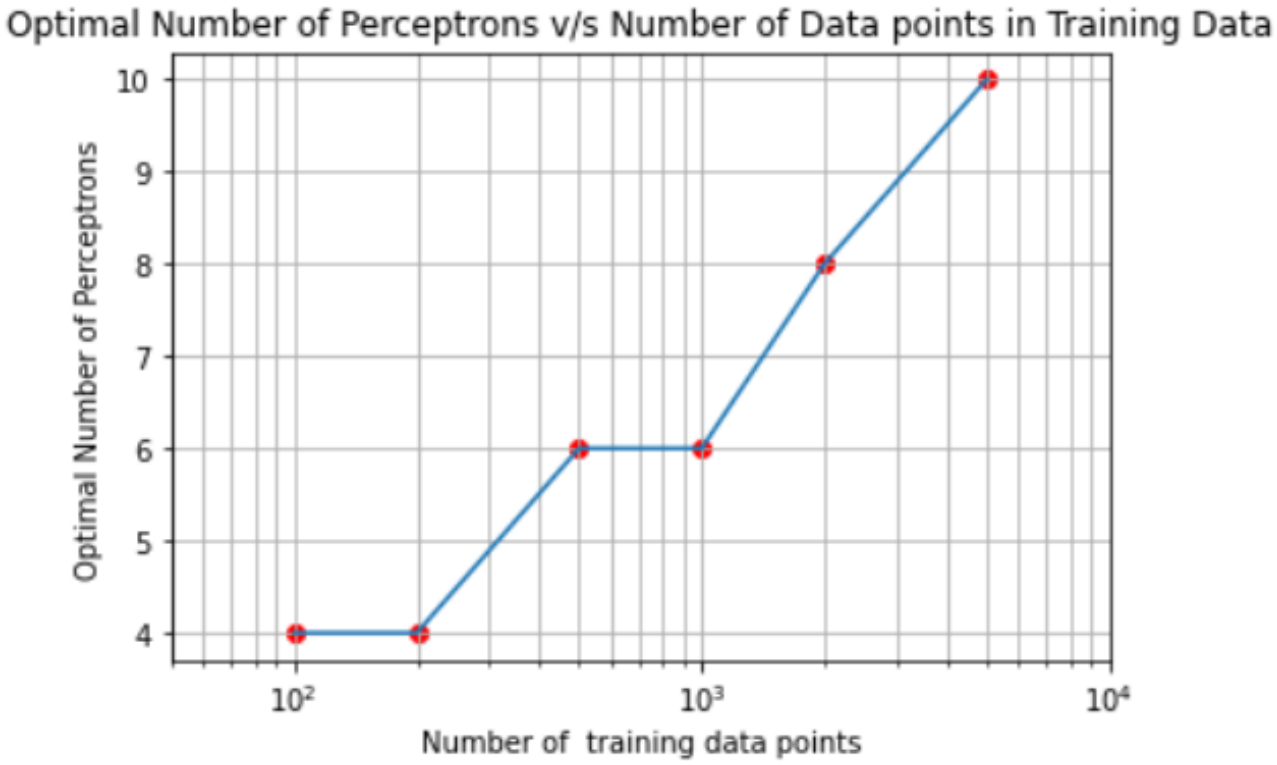
| Number of Input Samples | Optimum number of Perceptron's | Probability of Error (After doing performance analysis on 100k test dataset) |
|---|---|---|
| 100 | 4 | 0.37972 |
| 200 | 4 | 0.21622 |
| 500 | 6 | 0.19382 |
| 1000 | 6 | 0.19607 |
| 2000 | 8 | 0.176 |
| 5000 | 10 | 0.18476 |

From the table we can conclude that as the size of the number of input samples increases, the probability of error decreases.

Following graph shows the output of the usage of software packages elucidated above. The probability of error v/s number of data points in training data plot again explains that the error probability decreases as the size of the dataset begins to increase. (The straight orange line depicts the empirically estimated test error for the theoretically optimal classifier)



The following graph shows the Optimal number of perceptron's number against the Number of data points. The trend we can see here is that as the number of training data points are increased, the optimum number of perceptron's required increased as well.




CODE STARTS HERE:

```
#Importing all the required libraries

import random
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
from mpl_toolkits import mplot3d
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
```

```
from sklearn.model_selection import KFold

#Ignoring the output compiler warnings given while execution on Google Col
lab
import warnings
warnings.filterwarnings('ignore')
#-----------------------------------------

#Initializing the values of class means, covariances and class priors

#The class priors for all the 4 classes were uniform
p=0.25

#Means matrix
m0 = np.transpose(np.array([2.2,0,0]))
m1 = np.transpose(np.array([2.2,2.2,0]))
m2 = np.transpose(np.array([2.2,0,2.2]))
m3 = np.transpose(np.array([0,2.2,0]))

#Covariance matrix
c0 = np.array([[1,-0.5,0.3],[-0.5,1,-0.5],[0.3,-0.5,1]])
c1 = np.array([[1,0.5,0],[0.5,1,0.5],[0,0.5,1]])
c2 = np.array([[1,0.3,-0.2],[0.3,1,0.3],[-0.2,0.3,1]])
c3 = np.array([[1,0.3,0.6],[0.3,1,0.3],[0.6,0.3,1]])
#---------------------------------------------------------------

#Defining the MAP_loss and Risk calculation function, which will then be u
sed to check whether the mean, covariance taken lie under an error of 10 t
o 20%

def risk(i,x,lam_matrix):
  total = 0
  for j in range(4):
    total += lam_matrix[i][j]*p[j]*multivariate_normal.pdf(x,mean[j],cov[j
])
  return total

def MAP_loss(gauss,lam_matrix):
  for i in r[gauss]:
    choice = np.argmin([risk(0,i,lam_matrix),risk(1,i,lam_matrix),risk(2,i
,lam_matrix),risk(3,i,lam_matrix)])
    loss_matrix[gauss][choice] = loss_matrix[gauss][choice] + 1
  return loss_matrix
#---------------------------------------------------------------
```

```
#100 Samples
#Generation of datapoints for 100 samples
#Checking the probability of error (and then printing the error) in the me
ans and covariances when samples are 100

N=100
N_samples_100 = [0,0,0,0]
r = []

mean = [m0,m1,m2,m3]
cov = [c0,c1,c2,c3]
p = [0.25,0.25,0.25,0.25]

#Initializing the lambda and loss matrices, to be used in calculation of M
AP error
loss_matrix = [[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
lam_matrix = [[0,1,1,1],[1,0,1,1],[1,1,0,1],[1,1,1,0]]

#Sample Generation for Class labels 0, 1, 2 & 3
for i in range(N):
  a = random.random()
  if a<0.25:
    N_samples_100[0]+=1
  elif a<0.5:
    N_samples_100[1]+=1
  elif a<0.75:
    N_samples_100[2]+=1
  else:
    N_samples_100[3]+=1

#Storing generated samples in list 'r' declared previously, which are bein
g used for MAP_loss function
#for i in range(4):
  #r.append(np.random.multivariate_normal(mean[i],cov[i],N_samples_100[i])
)

#Generating random samples for all the 4 classes
r0_100 = np.random.multivariate_normal(m0,c0,N_samples_100[0])
r1_100 = np.random.multivariate_normal(m1,c1,N_samples_100[1])
r2_100 = np.random.multivariate_normal(m2,c2,N_samples_100[2])
r3_100 = np.random.multivariate_normal(m3,c3,N_samples_100[3])
r = [r0_100,r1_100,r2_100,r3_100]

#Calculating MAP to determine overall error obtained from the means and co
variances for all 4 classes (0,1,2,3)
```

```
MAP_loss(0,lam_matrix)
MAP_loss(1,lam_matrix)
MAP_loss(2,lam_matrix)
MAP_loss(3,lam_matrix)

total_loss = 0

#Printing of the total loss
for i in range(4):
  for j in range(4):
    if i!=j:
      total_loss += loss_matrix[i][j]
print(total_loss/N)
#-----------------------------------------------------------

#Plotting the 3-D scatter plot for 100 Training Samples


ax = plt.axes(projection ="3d")

# Creating plot
plt.title("simple 3D scatter plot")

plt.scatter(r0_100[:,0],r0_100[:,1],r0_100[:,2],c='b')
plt.scatter(r1_100[:,0],r1_100[:,1],r1_100[:,2],c='r')
plt.scatter(r2_100[:,0],r2_100[:,1],r2_100[:,2],c='g')
plt.scatter(r3_100[:,0],r3_100[:,1],r3_100[:,2],c='k')

#Adding title and labels for x,y,z axis
plt.title("D100 training sets")
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')

#Setting scatter plot limits
ax.set_xlim3d(-3,4)
ax.set_ylim3d(0,5.5)
ax.set_zlim3d(-0.1,0.1)

plt.show()
#-------------------------------------------------
#200 Samples
#Generation of datapoints for 200 samples
#Checking the probability of error (and then printing the error) in the me
ans and covariances when samples are 200
```

```
N=200
N_samples_200 = [0,0,0,0]
r = []

mean = [m0,m1,m2,m3]
cov = [c0,c1,c2,c3]
p = [0.25,0.25,0.25,0.25]

#Initializing the lambda and loss matrices, to be used in calculation of M
AP error
loss_matrix = [[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
lam_matrix = [[0,1,1,1],[1,0,1,1],[1,1,0,1],[1,1,1,0]]

#Sample generation for class labels 0,1,2 & 3
for i in range(N):
  a = random.random()
  if a<0.25:
    N_samples_200[0]+=1
  elif a<0.5:
    N_samples_200[1]+=1
  elif a<0.75:
    N_samples_200[2]+=1
  else:
    N_samples_200[3]+=1

r0_200 = np.random.multivariate_normal(m0,c0,N_samples_200[0])
r1_200 = np.random.multivariate_normal(m1,c1,N_samples_200[1])
r2_200 = np.random.multivariate_normal(m2,c2,N_samples_200[2])
r3_200 = np.random.multivariate_normal(m3,c3,N_samples_200[3])
r = [r0_200,r1_200,r2_200,r3_200]

#Calculating loss in the given means and covariance
MAP_loss(0,lam_matrix)
MAP_loss(1,lam_matrix)
MAP_loss(2,lam_matrix)
MAP_loss(3,lam_matrix)

total_loss = 0

#Showing output of total loss
for i in range(4):
  for j in range(4):
    if i!=j:
      total_loss += loss_matrix[i][j]
```

```
print(total_loss/N)
#-------------------------------------------------------------

#Plotting the 3-D scatter plot for 200 Training Samples

ax = plt.axes(projection ="3d")

# Creating plot
plt.title("simple 3D scatter plot")

plt.scatter(r0_200[:,0],r0_200[:,1],r0_200[:,2],c='b')
plt.scatter(r1_200[:,0],r1_200[:,1],r1_200[:,2],c='r')
plt.scatter(r2_200[:,0],r2_200[:,1],r2_200[:,2],c='g')
plt.scatter(r3_200[:,0],r3_200[:,1],r3_200[:,2],c='k')

#Adding scatter plot title and labels for the 3 different axis
plt.title("D200 training sets")
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')

#Setting the scatterplot limits
ax.set_xlim3d(-3,4)
ax.set_ylim3d(0,5.5)
ax.set_zlim3d(-0.1,0.1)

plt.show()
#-------------------------------------------------------------
#500 Samples
#Generation of datapoints for 500 samples
#Checking the probability of error (and then printing the error) in the me
ans and covariances when samples are 500

N=500
N_samples_500 = [0,0,0,0]
r = []

mean = [m0,m1,m2,m3]
cov = [c0,c1,c2,c3]
p = [0.25,0.25,0.25,0.25]

#Initializing the lambda and loss matrices, to be used in calculation of M
AP error
loss_matrix = [[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
lam_matrix = [[0,1,1,1],[1,0,1,1],[1,1,0,1],[1,1,1,0]]
```

```
#Generating random samples for classes 0, 1, 2 & 3
for i in range(N):
  a = random.random()
  if a<0.25:
    N_samples_500[0]+=1
  elif a<0.5:
    N_samples_500[1]+=1
  elif a<0.75:
    N_samples_500[2]+=1
  else:
    N_samples_500[3]+=1

r0_500 = np.random.multivariate_normal(m0,c0,N_samples_500[0])
r1_500 = np.random.multivariate_normal(m1,c1,N_samples_500[1])
r2_500 = np.random.multivariate_normal(m2,c2,N_samples_500[2])
r3_500 = np.random.multivariate_normal(m3,c3,N_samples_500[3])
r = [r0_500,r1_500,r2_500,r3_500]

#Determining MAP loss for each class (ie 0,1,2,3)
MAP_loss(0,lam_matrix)
MAP_loss(1,lam_matrix)
MAP_loss(2,lam_matrix)
MAP_loss(3,lam_matrix)

total_loss = 0

for i in range(4):
  for j in range(4):
    if i!=j:
      total_loss += loss_matrix[i][j]
print(total_loss/N)
#-------------------------------------------------------------

#Plotting the 3-D scatter plot for 500 Training Samples

ax = plt.axes(projection ="3d")

# Creating plot
plt.title("simple 3D scatter plot")

plt.scatter(r0_500[:,0],r0_500[:,1],r0_500[:,2],c='b')
plt.scatter(r1_500[:,0],r1_500[:,1],r1_500[:,2],c='r')
plt.scatter(r2_500[:,0],r2_500[:,1],r2_500[:,2],c='g')
plt.scatter(r3_500[:,0],r3_500[:,1],r3_500[:,2],c='k')
```

```
#Adding label and title
plt.title("D500 training sets")
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')

#Setting limits of the axes
ax.set_xlim3d(-3,4)
ax.set_ylim3d(0,5.5)
ax.set_zlim3d(-4,2)

plt.show()
#-----------------------------------------------------------------------

#1000 Samples
#Generation of datapoints for 1000 samples
#Checking the probability of error (and then printing the error) in the me
ans and covariances when samples are 1000

N=1000
N_samples_1k = [0,0,0,0]
r = []

mean = [m0,m1,m2,m3]
cov = [c0,c1,c2,c3]
p = [0.25,0.25,0.25,0.25]

#Initializing the lambda and loss matrices, to be used in calculation of M
AP error
loss_matrix = [[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
lam_matrix = [[0,1,1,1],[1,0,1,1],[1,1,0,1],[1,1,1,0]]

for i in range(N):
  a = random.random()
  if a<0.25:
    N_samples_1k[0]+=1
  elif a<0.5:
    N_samples_1k[1]+=1
  elif a<0.75:
    N_samples_1k[2]+=1
  else:
    N_samples_1k[3]+=1

r0_1k = np.random.multivariate_normal(m0,c0,N_samples_1k[0])
```

```
r1_1k = np.random.multivariate_normal(m1,c1,N_samples_1k[1])
r2_1k = np.random.multivariate_normal(m2,c2,N_samples_1k[2])
r3_1k = np.random.multivariate_normal(m3,c3,N_samples_1k[3])
r = [r0_1k,r1_1k,r2_1k,r3_1k]

MAP_loss(0,lam_matrix)
MAP_loss(1,lam_matrix)
MAP_loss(2,lam_matrix)
MAP_loss(3,lam_matrix)

total_loss = 0

for i in range(4):
  for j in range(4):
    if i!=j:
      total_loss += loss_matrix[i][j]
print(total_loss/N)
#---------------------------------------------------------------------

#Plotting the 3-D scatter plot for 1000 Training Samples

ax = plt.axes(projection ="3d")

# Creating plot
plt.title("simple 3D scatter plot")

#Creation of scatter plot
plt.scatter(r0_1k[:,0],r0_1k[:,1],r0_1k[:,2],c='b')
plt.scatter(r1_1k[:,0],r1_1k[:,1],r1_1k[:,2],c='r')
plt.scatter(r2_1k[:,0],r2_1k[:,1],r2_1k[:,2],c='g')
plt.scatter(r3_1k[:,0],r3_1k[:,1],r3_1k[:,2],c='k')

#Adding of title and labels of axes
plt.title("D1000 training sets")
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')

#Setting limits for the three axes
ax.set_xlim3d(-3,3.5)
ax.set_ylim3d(0,5.5)
ax.set_zlim3d(-4,2)

plt.show()
#-----------------------------------------------------
```

```
#2000 Samples
#Generation of datapoints for 2000 samples
#Checking the probability of error (and then printing the error) in the me
ans and covariances when samples are 2000

N=2000
N_samples_2k = [0,0,0,0]
r = []

mean = [m0,m1,m2,m3]
cov = [c0,c1,c2,c3]
p = [0.25,0.25,0.25,0.25]

#Loss & Lambda matrix will be used to determing error and check whether it
 lies between 10 and 20%
loss_matrix = [[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
lam_matrix = [[0,1,1,1],[1,0,1,1],[1,1,0,1],[1,1,1,0]]

for i in range(N):
  a = random.random()
  if a<0.25:
    N_samples_2k[0]+=1
  elif a<0.5:
    N_samples_2k[1]+=1
  elif a<0.75:
    N_samples_2k[2]+=1
  else:
    N_samples_2k[3]+=1

r0_2k = np.random.multivariate_normal(m0,c0,N_samples_2k[0])
r1_2k = np.random.multivariate_normal(m1,c1,N_samples_2k[1])
r2_2k = np.random.multivariate_normal(m2,c2,N_samples_2k[2])
r3_2k = np.random.multivariate_normal(m3,c3,N_samples_2k[3])
r = [r0_2k,r1_2k,r2_2k,r3_2k]

#Function call to return error obtained
MAP_loss(0,lam_matrix)
MAP_loss(1,lam_matrix)
MAP_loss(2,lam_matrix)
MAP_loss(3,lam_matrix)

total_loss = 0

#Printing total loss
```

```
for i in range(4):
  for j in range(4):
    if i!=j:
      total_loss += loss_matrix[i][j]
print(total_loss/N)
#--------------------------------------

#Plotting the 3-D scatter plot for 2000 Training Samples

ax = plt.axes(projection ="3d")

# Creating plot
plt.title("simple 3D scatter plot")

plt.scatter(r0_2k[:,0],r0_2k[:,1],r0_2k[:,2],c='b')
plt.scatter(r1_2k[:,0],r1_2k[:,1],r1_2k[:,2],c='r')
plt.scatter(r2_2k[:,0],r2_2k[:,1],r2_2k[:,2],c='g')
plt.scatter(r3_2k[:,0],r3_2k[:,1],r3_2k[:,2],c='k')

#Setting scatter plot title and axes labels
plt.title("D2000 training sets")
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')

#Setting the axes limits
ax.set_xlim3d(-3,4)
ax.set_ylim3d(0,5.5)
ax.set_zlim3d(-4,2)

plt.show()
#-----------------------------------------------------------

#5000 Samples
#Generation of datapoints for 5000 samples
#Checking the probability of error (and then printing the error) in the me
ans and covariances when samples are 5000

N=5000
N_samples_5k = [0,0,0,0]
r = []

mean = [m0,m1,m2,m3]
cov = [c0,c1,c2,c3]
p = [0.25,0.25,0.25,0.25]
```

```
#Loss and Lambda matrix are being used to determine error
loss_matrix = [[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
lam_matrix = [[0,1,1,1],[1,0,1,1],[1,1,0,1],[1,1,1,0]]

for i in range(N):
  a = random.random()
  if a<0.25:
    N_samples_5k[0]+=1
  elif a<0.5:
    N_samples_5k[1]+=1
  elif a<0.75:
    N_samples_5k[2]+=1
  else:
    N_samples_5k[3]+=1

r0_5k = np.random.multivariate_normal(m0,c0,N_samples_5k[0])
r1_5k = np.random.multivariate_normal(m1,c1,N_samples_5k[1])
r2_5k = np.random.multivariate_normal(m2,c2,N_samples_5k[2])
r3_5k = np.random.multivariate_normal(m3,c3,N_samples_5k[3])
r = [r0_5k,r1_5k,r2_5k,r3_5k]

MAP_loss(0,lam_matrix)
MAP_loss(1,lam_matrix)
MAP_loss(2,lam_matrix)
MAP_loss(3,lam_matrix)

total_loss = 0

#Printing out the error
for i in range(4):
  for j in range(4):
    if i!=j:
      total_loss += loss_matrix[i][j]
print(total_loss/N)
#-------------------------------------------------------------

#Plotting the 3-D scatter plot for 5000 Training Samples

ax = plt.axes(projection ="3d")

# Creating plot
plt.title("simple 3D scatter plot")

plt.scatter(r0_5k[:,0],r0_5k[:,1],r0_5k[:,2],c='b')
```

```
plt.scatter(r1_5k[:,0],r1_5k[:,1],r1_5k[:,2],c='r')
plt.scatter(r2_5k[:,0],r2_5k[:,1],r2_5k[:,2],c='g')
plt.scatter(r3_5k[:,0],r3_5k[:,1],r3_5k[:,2],c='k')

#Displaying title and axes labels
plt.title("D5000 training sets")
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')

ax.set_xlim3d(-3,5)
ax.set_ylim3d(0,5.5)
ax.set_zlim3d(-4,2)

plt.show()
#------------------------------------------------------------------

#100k Test Data Samples
#Generation of datapoints for 100000 test data samples
#Checking the probability of error (and then printing the error) in the me
ans and covariances when samples are 100k

N=100000
N_samples_100k = [0,0,0,0]
r = []

mean = [m0,m1,m2,m3]
cov = [c0,c1,c2,c3]
p = [0.25,0.25,0.25,0.25]
loss_matrix = [[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
lam_matrix = [[0,1,1,1],[1,0,1,1],[1,1,0,1],[1,1,1,0]]

for i in range(N):
  a = random.random()
  if a<0.25:
    N_samples_100k[0]+=1
  elif a<0.5:
    N_samples_100k[1]+=1
  elif a<0.75:
    N_samples_100k[2]+=1
  else:
    N_samples_100k[3]+=1

r0_100k = np.random.multivariate_normal(m0,c0,N_samples_100k[0])
r1_100k = np.random.multivariate_normal(m1,c1,N_samples_100k[1])
```

```
r2_100k = np.random.multivariate_normal(m2,c2,N_samples_100k[2])
r3_100k = np.random.multivariate_normal(m3,c3,N_samples_100k[3])
r = [r0_100k,r1_100k,r2_100k,r3_100k]

MAP_loss(0,lam_matrix)
MAP_loss(1,lam_matrix)
MAP_loss(2,lam_matrix)
MAP_loss(3,lam_matrix)

total_loss = 0

for i in range(4):
  for j in range(4):
    if i!=j:
      total_loss += loss_matrix[i][j]
print(total_loss/N)

error_theo = total_loss/N
#-----------------------------------------------------------------

#Plotting the 3-D scatter plot for 100000 Test Samples

ax = plt.axes(projection ="3d")

# Creating plot
plt.title("simple 3D scatter plot")

plt.scatter(r0_100k[:,0],r0_100k[:,1],r0_100k[:,2],c='b')
plt.scatter(r1_100k[:,0],r1_100k[:,1],r1_100k[:,2],c='r')
plt.scatter(r2_100k[:,0],r2_100k[:,1],r2_100k[:,2],c='g')
plt.scatter(r3_100k[:,0],r3_100k[:,1],r3_100k[:,2],c='k')

plt.title("D100k testing datasets")
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')

ax.set_xlim3d(-3,5)
ax.set_ylim3d(-1,5.5)
ax.set_zlim3d(-4,2)


plt.show()
#-----------------------------------------------------------------
#-----------------------------------------------------------------
```

```
#Model Order Selection

#Initializing empty list for storing the ideal number of perceptrons for e
ach training sample
perceptron_number = []

#Initializing empty list to store the probability of error for each traini
ng sample
prob_error = []

#Storing the perceptron values in a list to be used for MLP training
p_values = [4,6,8,10,12,14]

#These are the labels of the test dataset (100k Samples) which will be use
d on the trained MLP model of each train dataset
#The 4 different classes are concatenated into 1 array of dimension (10000
0,2)

r_100k = np.concatenate((r0_100k,r1_100k,r2_100k,r3_100k))
a_100k = np.array(np.zeros(N_samples_100k[0]))
b_100k = np.array(np.ones(N_samples_100k[1]))
c_100k = np.array(2*np.ones(N_samples_100k[2]))
d_100k = np.array(3*np.ones(N_samples_100k[3]))
r_100k_l = np.concatenate((a_100k,b_100k,c_100k,d_100k))
r_100k_labels = r_100k_l.reshape(100000,1)
#-----------------------------------------------------------------


#100 Samples
#These are the labels of the train dataset which contains 100 samples, all
 of them are concatenated into one list along with their class labels

r_100 = np.concatenate((r0_100,r1_100,r2_100,r3_100))
a_100 = np.array(np.zeros(N_samples_100[0]))
b_100 = np.array(np.ones(N_samples_100[1]))
c_100 = np.array(2*np.ones(N_samples_100[2]))
d_100 = np.array(3*np.ones(N_samples_100[3]))
r_100_l = np.concatenate((a_100,b_100,c_100,d_100))
r_100_labels = r_100_l.reshape(100,1)
#-----------------------------------------------------------------
#The optimal number of perceptrons are obtained via this loop with the hel
p of K fold cross validation and MLP classifier
```

```
#An empty list is created to store the accuracy, which will later be used
to obtain error
per_number = []

for p in p_values:
    kf = KFold(n_splits=10)
    per_acc = []
    for train_index, validation_index in kf.split(r_100):
      clf = MLPClassifier(random_state=1, max_iter=1000,hidden_layer_sizes
=(p,4),activation='relu').fit(r_100[train_index,:],r_100_labels[train_inde
x,:])
      clf.out_activation_ = 'softmax'
      acc = clf.score(r_100[validation_index,:],r_100_labels[validation_in
dex,:])
      per_acc.append(acc)
    per_number.append(sum(per_acc))
#-----------------------------------------------------------------

#The final optimal number of perceptrons are obtained, printed and then st
ored in the perceptron number array
final_p_100 =  p_values[np.argmin(np.array(per_number))]
print("Optimal number of Perceptrons: ",final_p_100)
perceptron_number.append(final_p_100)

#After the P number is obtained, the MLP Classifier is again trained with
the optimal number of perceptrons
#The accuracy (performance asssessment) is done by applying the trained mo
del on the test data of 100k samples
clf = MLPClassifier(random_state=1, max_iter=1000,hidden_layer_sizes=(fina
l_p_100,4),activation='relu').fit(r_100,r_100_labels)
acc_100 = clf.score(r_100k,r_100k_labels)
err_100  = 1 - acc_100
print("Error Probability after training the model and assessing performanc
e on test dataset: ",err_100)
prob_error.append(err_100)

#-----------------------------------------------------------------
#200 Samples
#These are the labels of the train dataset which contains 200 samples, all
 of them are concatenated into one list along with their class labels

r_200 = np.concatenate((r0_200,r1_200,r2_200,r3_200))
a_200 = np.array(np.zeros(N_samples_200[0]))
b_200 = np.array(np.ones(N_samples_200[1]))
c_200 = np.array(2*np.ones(N_samples_200[2]))
```

```
d_200 = np.array(3*np.ones(N_samples_200[3]))
r_200_l = np.concatenate((a_200,b_200,c_200,d_200))
r_200_labels = r_200_l.reshape(200,1)
#-------------------------------------------------------------

#The optimal number of perceptrons are obtained via this loop with the hel
p of K fold cross validation and MLP classifier

#An empty list is created to store the accuracy, which will later be used
to obtain error
per_number = []

for p in p_values:
    kf = KFold(n_splits=10)
    per_acc = []
    for train_index, validation_index in kf.split(r_200):
      clf = MLPClassifier(random_state=1, max_iter=1000,hidden_layer_sizes
=(p,4),activation='relu').fit(r_200[train_index,:],r_200_labels[train_inde
x,:])
      clf.out_activation_ = 'softmax'
      acc = clf.score(r_200[validation_index,:],r_200_labels[validation_in
dex,:])
      per_acc.append(acc)
    per_number.append(sum(per_acc))
#-------------------------------------------------------------

#The final optimal number of perceptrons are obtained, printed and then st
ored in the perceptron number array
final_p_200 =  p_values[np.argmin(np.array(per_number))]
print("Optimal number of Perceptrons: ",final_p_200)
perceptron_number.append(final_p_200)

#After the P number is obtained, the MLP Classifier is again trained with
the optimal number of perceptrons
#The accuracy (performance asssessment) is done by applying the trained mo
del on the test data of 100k samples
clf = MLPClassifier(random_state=1, max_iter=1000,hidden_layer_sizes=(fina
l_p_200,4),activation='relu').fit(r_200,r_200_labels)
acc_200 = clf.score(r_100k,r_100k_labels)
err_200  = 1 - acc_200
print("Error Probability after training the model and assessing performanc
e on test dataset: ",err_200)
prob_error.append(err_200)
#-------------------------------------------------------------
```

```
#500 Samples
#These are the labels of the train dataset which contains 500 samples, all
 of them are concatenated into one list along with their class labels

r_500 = np.concatenate((r0_500,r1_500,r2_500,r3_500))
a_500 = np.array(np.zeros(N_samples_500[0]))
b_500 = np.array(np.ones(N_samples_500[1]))
c_500 = np.array(2*np.ones(N_samples_500[2]))
d_500 = np.array(3*np.ones(N_samples_500[3]))
r_500_l = np.concatenate((a_500,b_500,c_500,d_500))
r_500_labels = r_500_l.reshape(500,1)
#-----------------------------------------------------------------

#The optimal number of perceptrons are obtained via this loop with the hel
p of K fold cross validation and MLP classifier

#An empty list is created to store the accuracy, which will later be used
to obtain error
per_number = []

for p in p_values:
    kf = KFold(n_splits=10)
    per_acc = []
    for train_index, validation_index in kf.split(r_500):
      clf = MLPClassifier(random_state=1, max_iter=1000,hidden_layer_sizes
=(p,4),activation='relu').fit(r_500[train_index,:],r_500_labels[train_inde
x,:])
      clf.out_activation_ = 'softmax'
      acc = clf.score(r_500[validation_index,:],r_500_labels[validation_in
dex,:])
      per_acc.append(acc)
    per_number.append(sum(per_acc))
#-----------------------------------------------------------------

#The final optimal number of perceptrons are obtained, printed and then st
ored in the perceptron number array
final_p_500 =  p_values[np.argmin(np.array(per_number))]
print("Optimal number of Perceptrons: ",final_p_500)
perceptron_number.append(final_p_500)

#After the P number is obtained, the MLP Classifier is again trained with
the optimal number of perceptrons
#The accuracy (performance asssessment) is done by applying the trained mo
del on the test data of 100k samples
```

```
clf = MLPClassifier(random_state=1, max_iter=1000,hidden_layer_sizes=(fina
l_p_500,4),activation='relu').fit(r_500,r_500_labels)
acc_500 = clf.score(r_100k,r_100k_labels)
err_500  = 1 - acc_500
print("Error Probability after training the model and assessing performanc
e on test dataset: ",err_500)
prob_error.append(err_500)
#---------------------------------------------------------------

#1000 Samples
#These are the labels of the train dataset which contains 1k samples, all
of them are concatenated into one list along with their class labels

r_1k = np.concatenate((r0_1k,r1_1k,r2_1k,r3_1k))
a_1k = np.array(np.zeros(N_samples_1k[0]))
b_1k = np.array(np.ones(N_samples_1k[1]))
c_1k = np.array(2*np.ones(N_samples_1k[2]))
d_1k = np.array(3*np.ones(N_samples_1k[3]))
r_1k_l = np.concatenate((a_1k,b_1k,c_1k,d_1k))
r_1k_labels = r_1k_l.reshape(1000,1)
#---------------------------------------------------------------

#The optimal number of perceptrons are obtained via this loop with the hel
p of K fold cross validation and MLP classifier

#An empty list is created to store the accuracy, which will later be used
to obtain error
per_number = []

for p in p_values:
    kf = KFold(n_splits=10)
    per_acc = []
    for train_index, validation_index in kf.split(r_1k):
      clf = MLPClassifier(random_state=1, max_iter=1000,hidden_layer_sizes
=(p,4),activation='relu').fit(r_1k[train_index,:],r_1k_labels[train_index,
:])
      clf.out_activation_ = 'softmax'
      acc = clf.score(r_1k[validation_index,:],r_1k_labels[validation_inde
x,:])
      per_acc.append(acc)
    per_number.append(sum(per_acc))
#---------------------------------------------------------------

#The final optimal number of perceptrons are obtained, printed and then st
ored in the perceptron number array
final_p_1k =  p_values[np.argmin(np.array(per_number))]
```

```
print("Optimal number of Perceptrons: ",final_p_1k)
perceptron_number.append(final_p_1k)

#After the P number is obtained, the MLP Classifier is again trained with
the optimal number of perceptrons
#The accuracy (performance asssessment) is done by applying the trained mo
del on the test data of 100k samples
clf = MLPClassifier(random_state=1, max_iter=1000,hidden_layer_sizes=(fina
l_p_1k,4),activation='relu').fit(r_1k,r_1k_labels)
acc_1k = clf.score(r_100k,r_100k_labels)
err_1k  = 1 - acc_1k
print("Error Probability after training the model and assessing performanc
e on test dataset: ",err_1k)
prob_error.append(err_1k)
#-----------------------------------------------------------------


#2000 Samples
#These are the labels of the train dataset which contains 2k samples, all
of them are concatenated into one list along with their class labels

r_2k = np.concatenate((r0_2k,r1_2k,r2_2k,r3_2k))
a_2k = np.array(np.zeros(N_samples_2k[0]))
b_2k = np.array(np.ones(N_samples_2k[1]))
c_2k = np.array(2*np.ones(N_samples_2k[2]))
d_2k = np.array(3*np.ones(N_samples_2k[3]))
r_2k_l = np.concatenate((a_2k,b_2k,c_2k,d_2k))
r_2k_labels = r_2k_l.reshape(2000,1)
#-----------------------------------------------------------------


#The optimal number of perceptrons are obtained via this loop with the hel
p of K fold cross validation and MLP classifier

#An empty list is created to store the accuracy, which will later be used
to obtain error
per_number = []

for p in p_values:
    kf = KFold(n_splits=10)
    per_acc = []
    for train_index, validation_index in kf.split(r_2k):
      clf = MLPClassifier(random_state=1, max_iter=1000,hidden_layer_sizes
=(p,4),activation='relu').fit(r_2k[train_index,:],r_2k_labels[train_index,
:])
      clf.out_activation_ = 'softmax'
```

```
      acc = clf.score(r_2k[validation_index,:],r_2k_labels[validation_inde
x,:])
      per_acc.append(acc)
    per_number.append(sum(per_acc))

#-----------------------------------------------------------------

#The final optimal number of perceptrons are obtained, printed and then st
ored in the perceptron number array
final_p_2k =  p_values[np.argmin(np.array(per_number))]
print("Optimal number of Perceptrons: ",final_p_2k)
perceptron_number.append(final_p_2k)

#After the P number is obtained, the MLP Classifier is again trained with
the optimal number of perceptrons
#The accuracy (performance asssessment) is done by applying the trained mo
del on the test data of 100k samples
clf = MLPClassifier(random_state=1, max_iter=1000,hidden_layer_sizes=(fina
l_p_2k,4),activation='relu').fit(r_2k,r_2k_labels)
acc_2k = clf.score(r_2k,r_2k_labels)
err_2k  = 1 - acc_2k
print("Error Probability after training the model and assessing performanc
e on test dataset: ",err_2k)
prob_error.append(err_2k)

#----------------------------------------
#5000 Samples
#These are the labels of the train dataset which contains 5k samples, all
of them are concatenated into one list along with their class labels

r_5k = np.concatenate((r0_5k,r1_5k,r2_5k,r3_5k))
a_5k = np.array(np.zeros(N_samples_5k[0]))
b_5k = np.array(np.ones(N_samples_5k[1]))
c_5k = np.array(2*np.ones(N_samples_5k[2]))
d_5k = np.array(3*np.ones(N_samples_5k[3]))
r_5k_l = np.concatenate((a_5k,b_5k,c_5k,d_5k))
r_5k_labels = r_5k_l.reshape(5000,1)

#----------------------------------------
#The optimal number of perceptrons are obtained via this loop with the hel
p of K fold cross validation and MLP classifier

#An empty list is created to store the accuracy, which will later be used
to obtain error
per_number = []
```

```
for p in p_values:
    kf = KFold(n_splits=10)
    per_acc = []
    for train_index, validation_index in kf.split(r_5k):
      clf = MLPClassifier(random_state=1, max_iter=1000,hidden_layer_sizes
=(p,4),activation='relu').fit(r_5k[train_index,:],r_5k_labels[train_index,
:])
      clf.out_activation_ = 'softmax'
      acc = clf.score(r_5k[validation_index,:],r_5k_labels[validation_inde
x,:])
      per_acc.append(acc)
    per_number.append(sum(per_acc))

#----------------------------------------
#The final optimal number of perceptrons are obtained, printed and then st
ored in the perceptron number array
final_p_5k =  p_values[np.argmin(np.array(per_number))]
print("Optimal number of Perceptrons: ",final_p_5k)
perceptron_number.append(final_p_5k)

clf = MLPClassifier(random_state=1, max_iter=1000,hidden_layer_sizes=(fina
l_p_5k,4),activation='relu').fit(r_5k,r_5k_labels)
acc_5k = clf.score(r_100k,r_100k_labels)
err_5k  = 1 - acc_5k
print("Error Probability after training the model and assessing performanc
e on test dataset: ",err_5k)
prob_error.append(err_5k)

#-------------------------------------------------------------
#Displaying lists formed by perceptron number (Optimum) & Error Probabilit
y
print("The ideal perceptron numbers for all training datasets is: ",percep
tron_number)
print("The corresponding error probability is: ",prob_error)
#-------------------------------------------------------------

#Plotting graph for Probability of Error v/s Number of Data points in Trai
ning Data
data_points = [100,200,500,1000,2000,5000]
plt.grid(True, which='both')
plt.semilogx(data_points,prob_error)
plt.scatter(data_points,prob_error,c='r')
plt.xlabel("Number of  training data points")
plt.ylabel("Probability of Error")
```

```
plt.title("Probability of Error v/s Number of Data points in Training Data
")
plt.plot([50,10000],[error_theo,error_theo])
plt.xlim(50,10000)
plt.show()
#-------------------------------------------------------

#Plotting graph for Optimal Number of Perceptrons v/s Number of Data point
s in Training Data
data_points = [100,200,500,1000,2000,5000]
plt.grid(True, which='both')
plt.semilogx(data_points,perceptron_number)
plt.scatter(data_points,perceptron_number,c='r')
plt.xlabel("Number of  training data points")
plt.ylabel("Optimal Number of Perceptrons")
plt.title("Optimal Number of Perceptrons v/s Number of Data points in Trai
ning Data")
plt.xlim(50,10000)
plt.show()

#--------------- END OF SOLUTION 1 ----------------------------------
```

I have implemented the above code using Google Collab, on the following URL:
https://colab.research.google.com/drive/1dPcA6j4fSqs8yoXjQ84Gtr0Yfuh5GRb6?usp=sharing