EECE 5644: Machine Learning Project 4 Report

1. Given the following data:

$$Priors = [0.5, 0.5]$$
$$number\ of\ samples = [1000, 10000]$$
$$\mathbf{x} = r_l \begin{bmatrix} cos(\theta) \\ sin(\theta) \end{bmatrix} + \mathbf{n}$$

   where, $\theta \sim$ uniform $[-\pi, \pi]$, n $\sim$N(0,$\sigma^2 I$), $r_{-1} = 2, r_{+1} = 4$,$\sigma = 1$

   Actual Data Distribution:

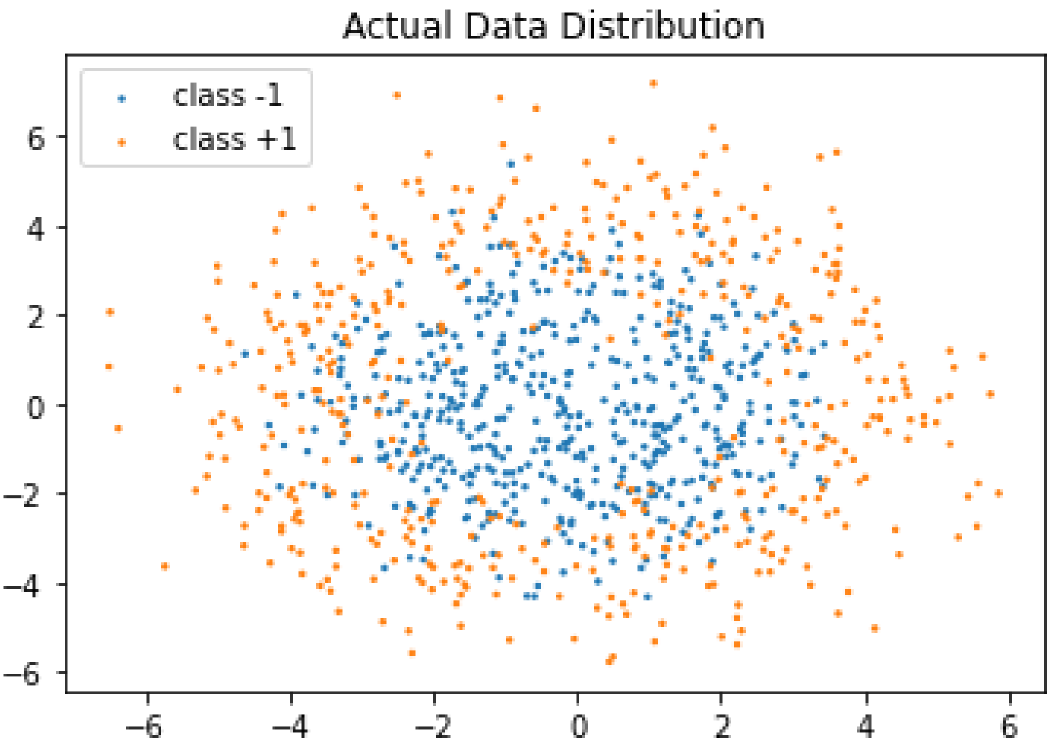


Figure 1: Actual training data distribution

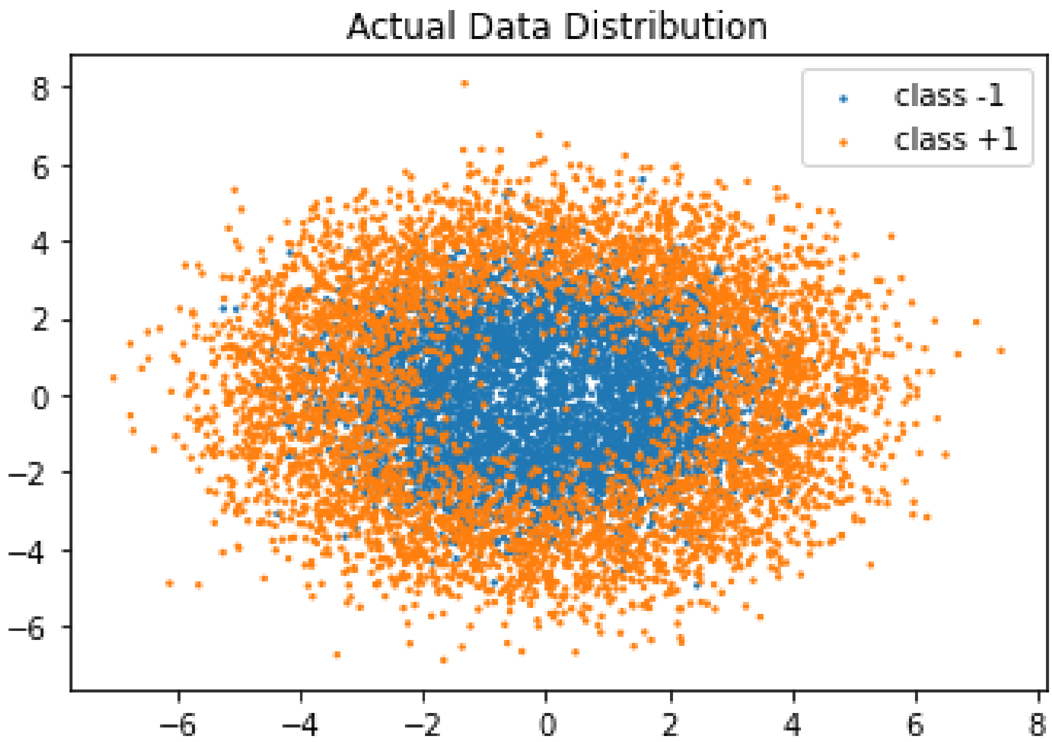


Figure 2: Actual test data distribution

I MLP

Implementation:

After generating the samples based on the above expression ,and Gaussian parameters (zero mean and identity covariance), I used the training dataset (1000 samples) for choosing the hyper-parameters and later apply MLP on the test dataset.

10-fold cross validation was used in order to select the optimal number of perceptrons,optimal activation function type for the hidden layer ,optimal number of epochs and optimizer type. To find the optimal number of epochs, I used a log-space function with base 5 and maximum limit of 625. Activation functions such as $ReLU, sigmoid,\ and\ tanh$ were compared with their minimum probability of error. After choosing the right epoch number and activation function type, I apply the same procedure to find the optimal number of perceptrons with the selected epoch number and activation function type, and the perceptron with the minimum error was selected. Similarly, teh optimizers were compared while I was choosing the number of perceptrons. Since the number of class is 2, I used a $Sigmoid$ activation function at the output layer.Using the hyper-parameters I got from the above procedure, the classifier was applied on the test dataset (10K), and the confusion matrix along with the error percentage and classification performance (with a threshold value of 0.5) plot with decision boundary super-imposed were reported.

a Number of epochs

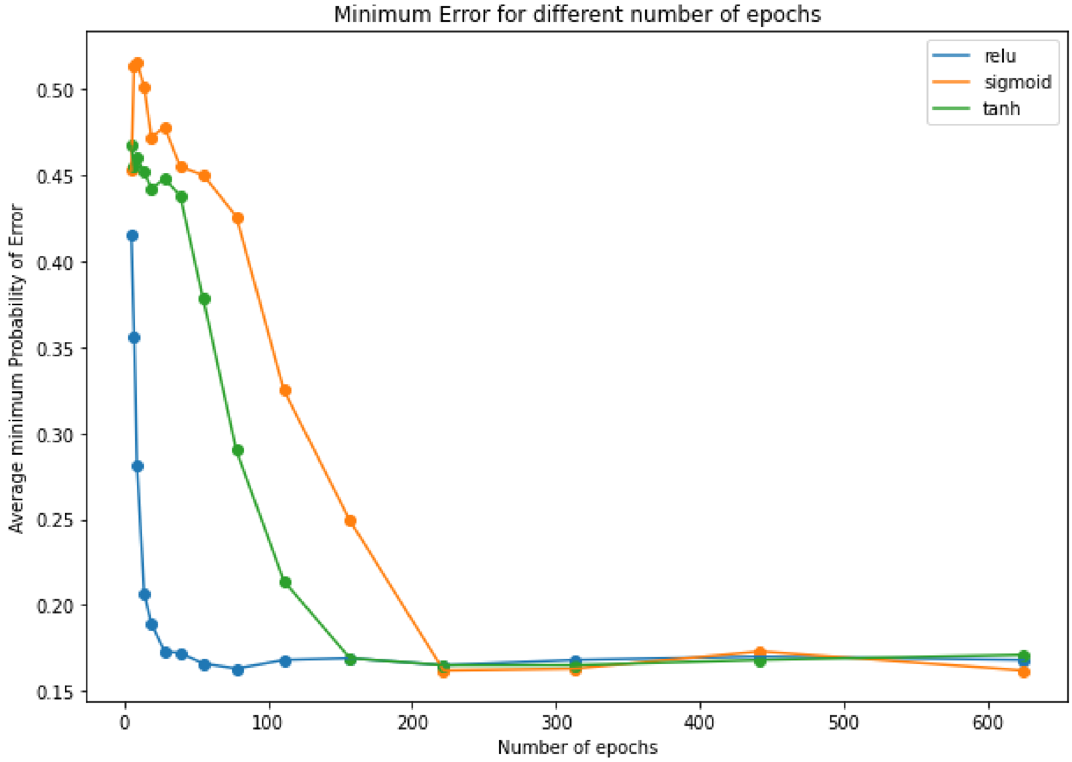


Figure 3: Minimum Error for different number of epochs

b Number of Perceptrons

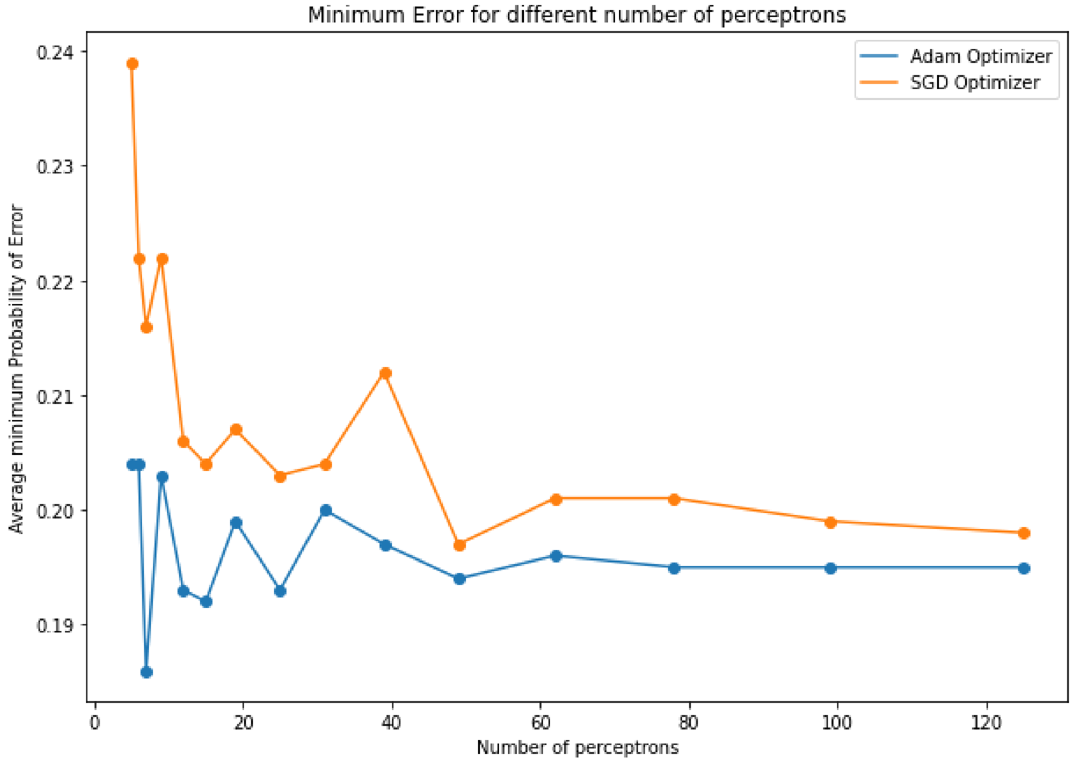


Figure 4: Minimum Error for different number of neurons under ReLU activation function

Minimum probability of error for Adam:0.186
Minimum probability of error for SGD:0.197

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	32
Number of perceptron	7
Optimizer	Adam
Activation function for hidden layer	ReLU
Activation function for output layer	Sigmoid
loss function	binary cross_entropy

c Confusion Matrix

The test dataset (10K), 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: Normalized Confusion matrix:

$$\begin{bmatrix} 4066 & 950 \\ 681 & 4303 \end{bmatrix} \qquad \begin{bmatrix} 0.81 & 0.19 \\ 0.14 & 0.86 \end{bmatrix}$$

Minimum probability of error: 0.163
Accuracy: 83.7%
cross_entropy loss:0.402

d Classification Performance:

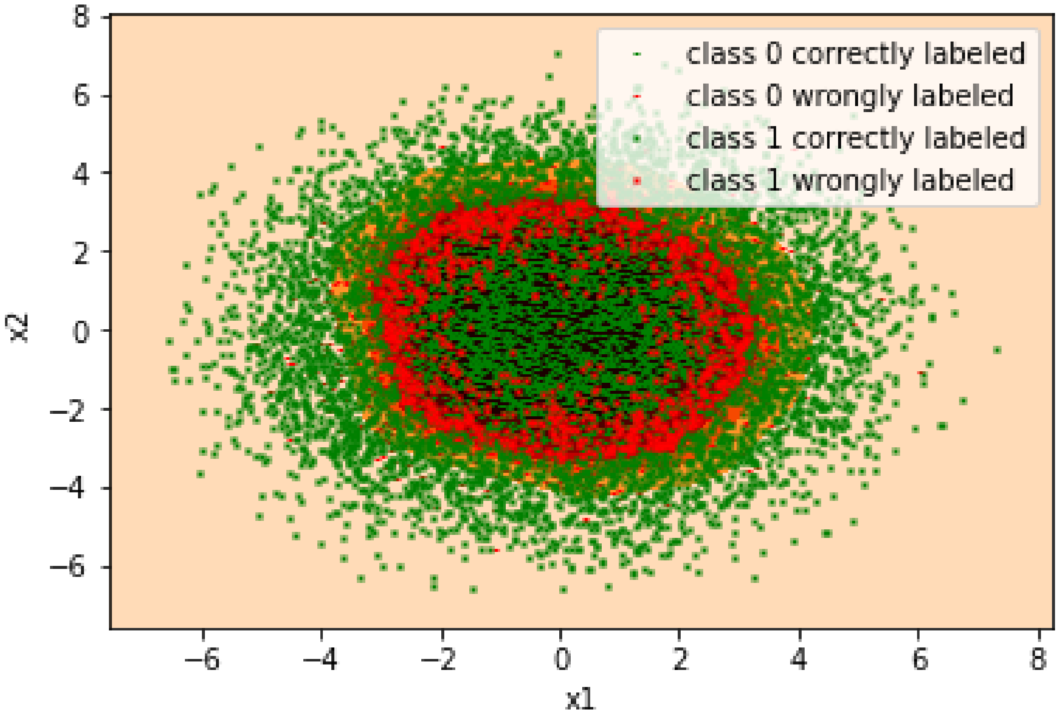


Figure 5: MLP Classification performace

II SVM

Implementation: The same data as above was used for this classifier, and 10-fold cross validation was performed on the training dataset to choose the hyper-parameters(C , γ) from the following combinations, C=[1000,100,10,1.0,0.1,0.01] and γ=[0.01,0.1,1.0,10,100,1000]. For choosing the optimal hyper-parameters, I used *GridSearchCV* function from sklearn library, and a combination of C=100 and γ=0.01 has highest accuracy (84.1%) and applied on the test dataset. A classification performance (with threshold value of 0.5) was plotted at the end, which shows the correct and incorrect labels with the decision boundary superimposed.

a Hyper-parameter Selection

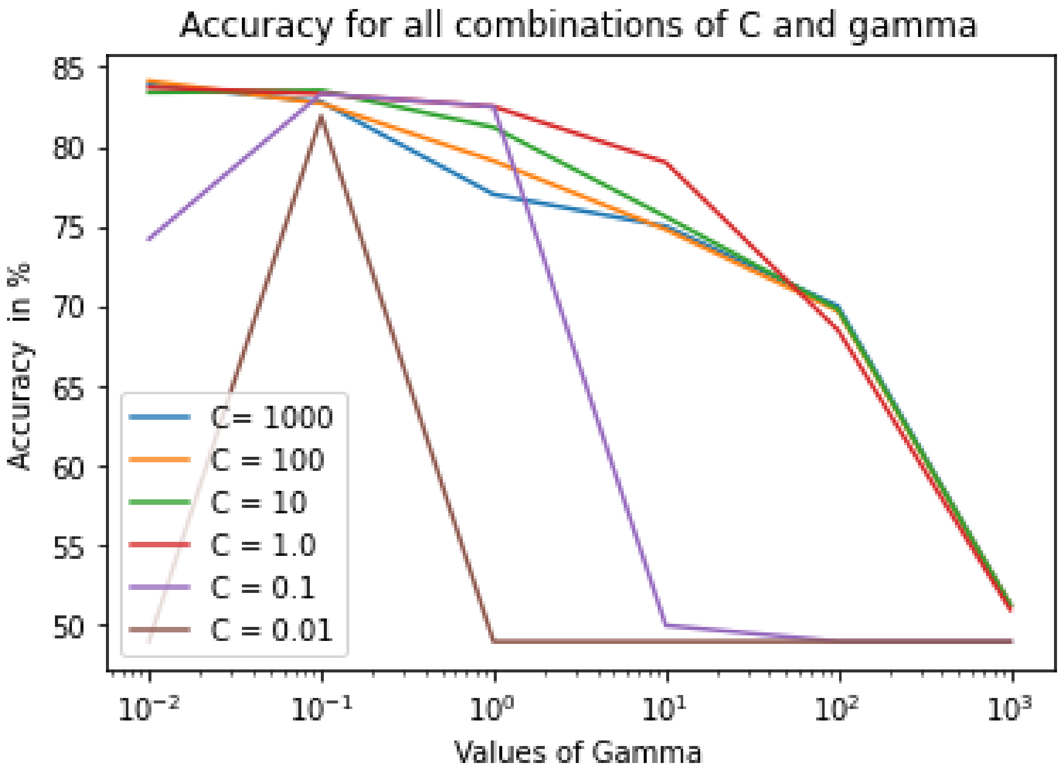


Figure 6: Accuracy for different combinations of hyper-parameters

Based on the above plots the following table is prepared and shows the selected parameter values (with the highest accuracy) for the test dataset.

C	100
γ	0.01
kernel	rbf

b Confusion Matrix
The test dataset (10K), 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:

$$\begin{bmatrix} 4202 & 841 \\ 822 & 4135 \end{bmatrix}$$

Normalized Confusion matrix:

$$\begin{bmatrix} 0.83 & 0.17 \\ 0.16 & 0.83 \end{bmatrix}$$

Minimum probability of error: 0.166
Accuracy: 83.4%

c Classification Performance:

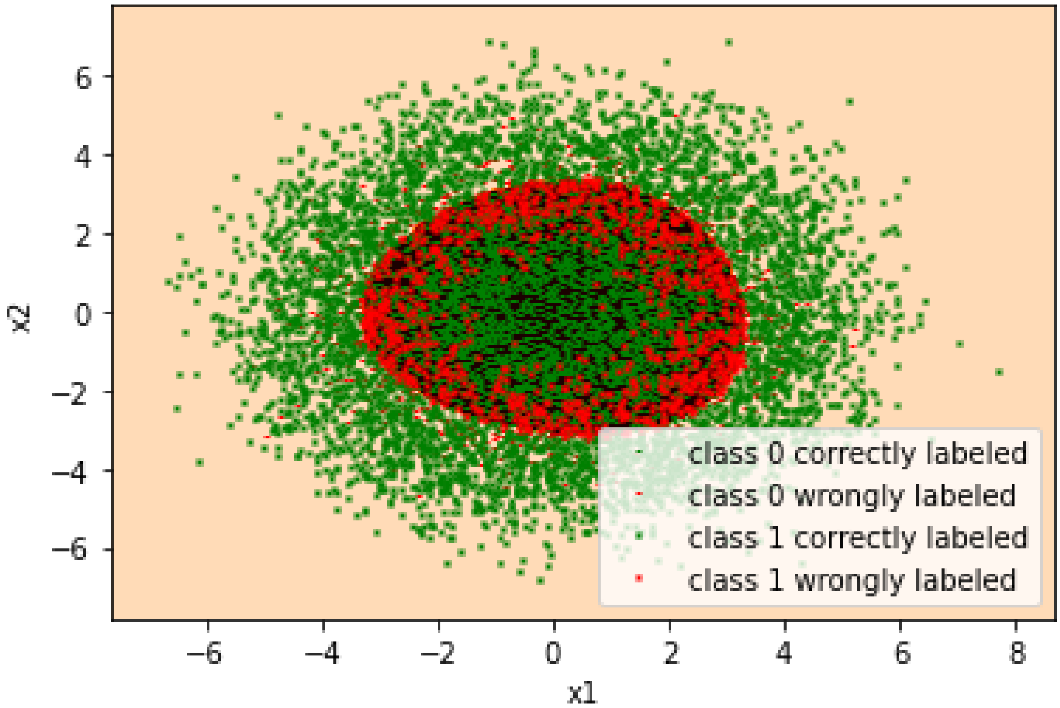


Figure 7: SVM Classification performance

III Conclusion

A comparison between the two classifiers is summarized with the below table.

Classifier	Training Accuracy (%)	Test Accuracy(%)
MLP	81.4	83.7
SVM	84.1	83.4

From the above table, we can see that the training accuracy for the SVM classifier is higher than the MLP, while for the test dataset the classification accuracy is almost equal. As the number of samples increases (from 1000 to 10k), there will be more samples in the decision boundary and as a result the SVM performs poor as the samples increase. While for MLP, the accuracy increases as samples increases because it will capture the true distribution of data and this will minimize both bias and variance.

2. GMM

Implementation:

I used three color images (deer,street and Penguin), from the Berkeley Segmentation Dataset and generate 5 features using the row index, column index ,red,green and blues pixel values. After forming the feature vector and scaled it to $[0, 1]$ using *min_maxscale* function from sklearn library, I used the built in function *GaussianMixture* from sklearn library 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 sklearn library 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).

I Image 1 (Deer)

a Model order selection

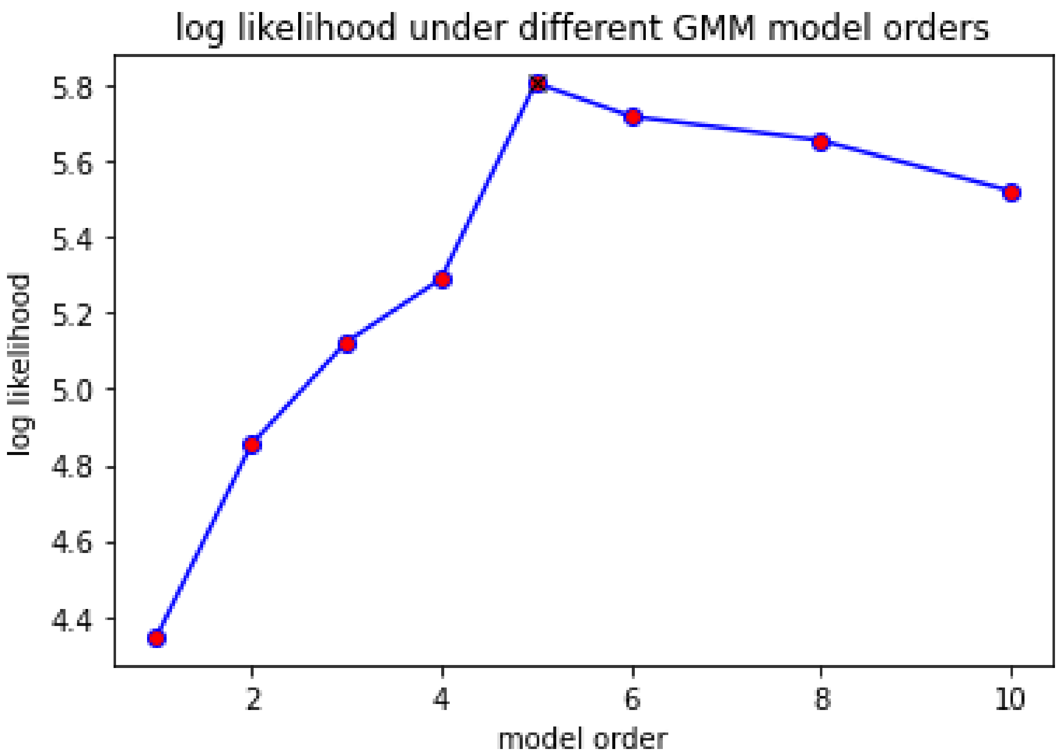
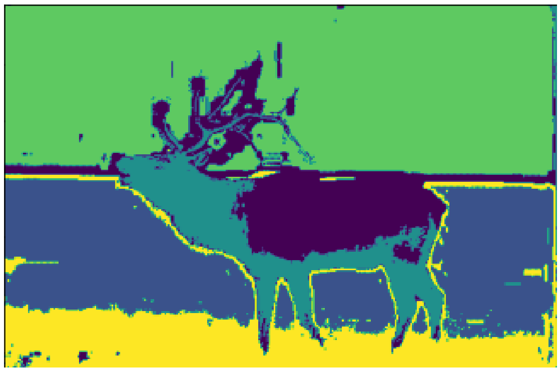


Figure 8: log-likelihood score for different clusters (model order)

As it can be seen from the above figure, a model order of 5 has the highest log liklihood score and was selected for the Gaussian component while segmenting the input image.

b Results

(a) Original image



(b) Segmented image

II Image 2(street)

a Model order selection

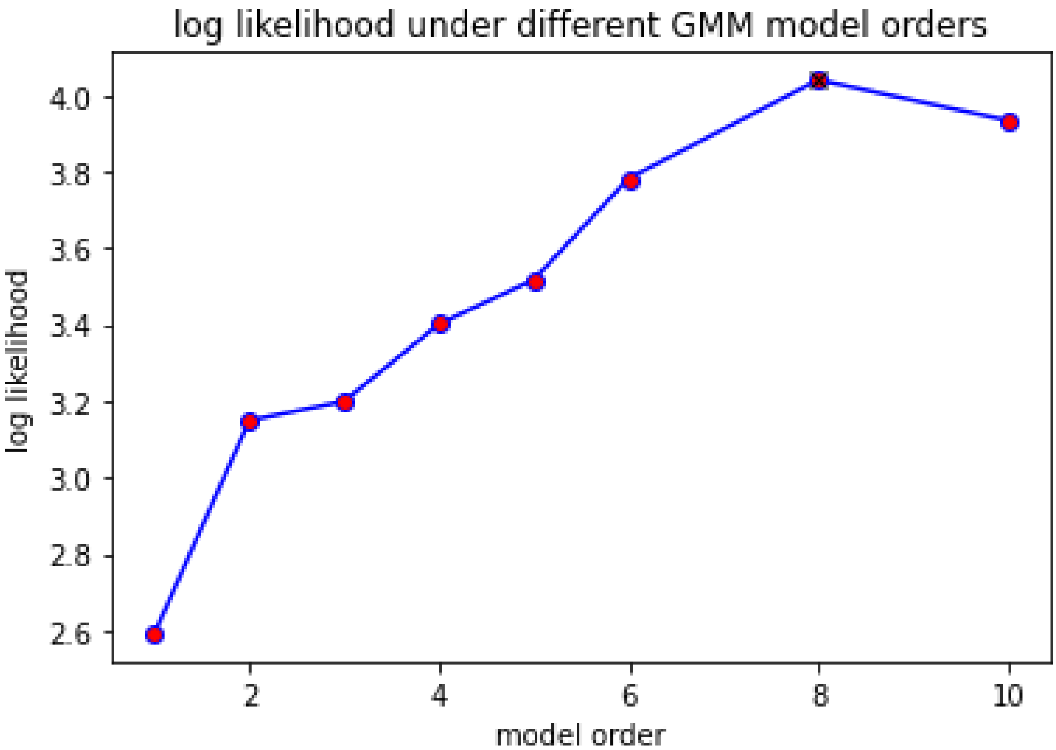
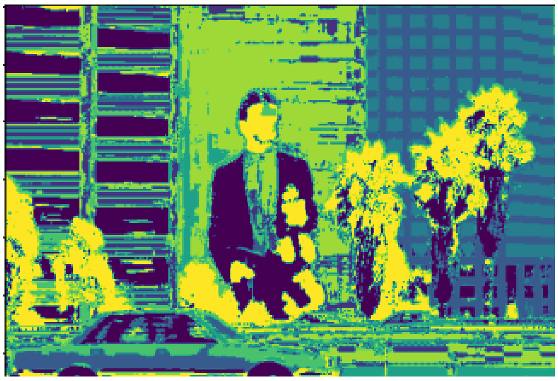


Figure 10: log-likelihood score for different clusters (model order)

As it can be seen from the above figure, a model order of 8 has the highest log liklihood score and was selected for the Gaussian component while segmenting the input image.

b Results

(a) Original image



(b) Segmented image

III Image 3 (Penguin)

a Model order selection

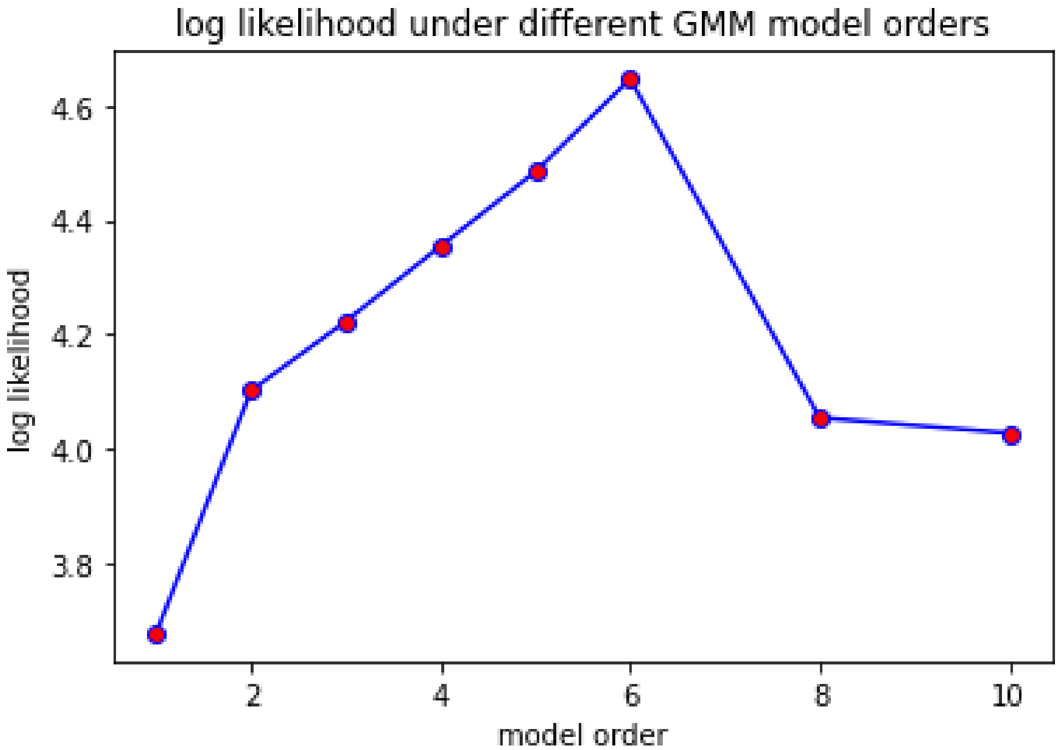
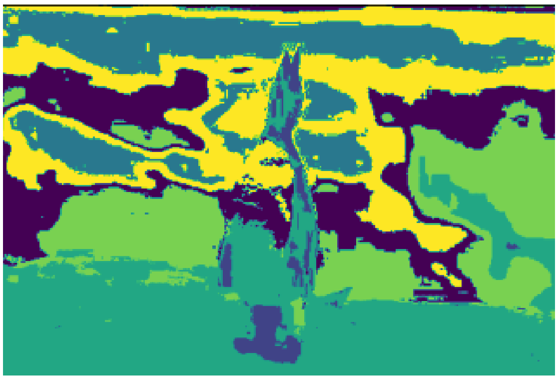


Figure 12: log-likelihood score for different clusters (model order)

As it can be seen from the above figure, a model order of 6 has the highest log liklihood score and was selected for the Gaussian component while segmenting the input image.

b Results

(a) Original image



(b) Segmented image

A Appendix

```
# Question 1
import numpy as np
from scipy . stats import multivariate_normal as mvn
import tensorflow as tf
from sklearn.metrics import confusion_matrix
import random
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)
import math

from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV,KFold
from sklearn.metrics import classification_report


def data(n):
  label = np.zeros((1,n))
  for i in range(n):
    label[0,i]= np.random.choice([0,1],p=prior) # equal prior

  x = np.zeros ((features , n))

  for index in range (n) :
    theta=np.random.uniform(theta_margin[0],theta_margin[1])
    temp=np.array([np.cos(theta),np.sin(theta)])

    if label [0,index] == 0:#sample from class -1
      x [:,index] = 2*(temp)+mvn(m1_n,c1_n).rvs(1)
    else:# from class +1
      x [:,index] = 4*(temp)+mvn(m2_n,c2_n).rvs(1)

  return x,label


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))
```

```python
  train=np.array(folds)

  xtest=test[:,0:2]
  ytest=test[:,2]
  xtrain=train[:,:,0:2].reshape(-1,2)
  ytrain=train[:,:,2].reshape(-1)

  return xtrain,ytrain,xtest,ytest

def MLP(sample_type):

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

  data_concat=np.hstack((X_train,Y_train))

  for activ in range(len(activations)):

    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)
        input_shape = (features,)
        Y_train_encod=to_categorical(Y_train_,num_class)
        model = tf.keras.models.Sequential()

        model.add(Dense(percpt,kernel_initializer='random_uniform',input_shape=input_shape, activation=activations[activ]))
        model.add(Dense(num_class, kernel_initializer='random_uniform',activation='softmax'))

        model.compile( optimizer='Adam',loss='categorical_crossentropy', metrics=['accuracy'])
        model.fit(X_train_,Y_train_encod, epochs=100, batch_size=32,verbose=0)
        temp_y=model.predict(X_test_)
        y_pred_=np.argmax(temp_y, axis=1)
        tmt=to_categorical(Y_test_,num_class)
        y_test=np.argmax(tmt, axis=1)

        cm = confusion_matrix(y_test, y_pred_,labels=[0.0,1.0])
        temp=(cm[0,0]+cm[1,1])/cm.sum()
        err_fold.append(1-temp)
        acc_fold.append(temp)

      error_mat[idx][activ]=(np.mean(err_fold)) # choose the perceptron based on the min of error
      accuracy_mat[idx][activ]=(np.mean(acc_fold))

   # find the perceptron with minimum error or loss
  opt_percerptron=np.where(error_mat==error_mat.flat[np.argmin(error_mat)])
  optimal_perceptron=percept_list[int(opt_percerptron[0][0])]

  print(f'optimal number of neurons:{optimal_perceptron}')
```

```python

  input_shape = (features,)
  print(f'Feature shape: {input_shape}')

  model = tf.keras.Sequential()
  model.add(Dense(optimal_perceptron,kernel_initializer='random_uniform', input_shape=input_shape, activation='relu'))
  model.add(Dense(1, kernel_initializer='random_uniform',activation='sigmoid'))
  model.compile(loss='binary_crossentropy', optimizer='Adam', metrics=['accuracy'])
  model.fit(X_train, Y_train, epochs=100, batch_size=32, verbose=1)

  # # Test the model after training
  test_results = model.evaluate(X_test, Y_test, verbose=1)
  print(f'Test results - Loss: {test_results[0]} - Accuracy: {test_results[1]}')
  p_pred = model.predict(X_test)
  p_pred = p_pred.flatten()
  # extract the predicted class labels
  y_pred = np.where(p_pred > 0.5, 1, 0)
  cm=confusion_matrix(Y_test, y_pred)
  normalized_cm=cm / cm.astype(float).sum(axis=1)
  error = (1- (cm[0,0]+cm[1,1])/cm.sum())

  print(f'minimum probability of error:{error}')
  print(f'confusion matrix:{cm}')
  print(f'Normalized confusion matrix:{normalized_cm}')


  min1, max1 = X_test[:, 0].min()-1, X_test[:, 0].max()+1
  min2, max2 = X_test[:, 1].min()-1, X_test[:, 1].max()+1
  # define the x and y scale
  x1grid = np.arange(min1, max1, 0.1)
  x2grid = np.arange(min2, max2, 0.1)

  xx, yy = np.meshgrid(x1grid, x2grid)

  r1, r2 = xx.flatten(), yy.flatten()
  r1, r2 = r1.reshape((len(r1), 1)), r2.reshape((len(r2), 1))
  # horizontal stack vectors to create x1,x2 input for the model
  grid = np.hstack((r1,r2))
  yhat = model.predict(grid)
  contour_val=[xx,yy,yhat]
  return error_mat,contour_val,y_pred

def SVM(sample_type):

  grid = dict(kernel=kernel,C=C_list,gamma= gamma_list )
  cross_val = KFold(n_splits=10)
  grid_object = GridSearchCV(estimator=SVC(), param_grid=grid, cv=cross_val, scoring='accuracy')
  grid_result = grid_object.fit(X_train, Y_train.reshape(-1))
  print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))

  svc = SVC(kernel = 'rbf', C = 100, gamma=0.01 )

```

```
  svc.fit(X_train,Y_train)
  y_pred=svc.predict(X_test)

  cm=confusion_matrix(Y_test, y_pred)
  normalized_cm=cm / cm.astype(float).sum(axis=1)
  error = (1- (cm[0,0]+cm[1,1])/cm.sum())

  print(classification_report(Y_test, y_pred))
  print('Model accuracy score : {0:0.4f}'. format(accuracy_score(Y_test,
    y_pred)))

  grid_temp=list(grid_result.cv_results_['mean_test_score']*100)

  min1, max1 = X_test[:, 0].min()-1, X_test[:, 0].max()+1
  min2, max2 = X_test[:, 1].min()-1, X_test[:, 1].max()+1

  x1grid = np.arange(min1, max1, 0.1)
  x2grid = np.arange(min2, max2, 0.1)

  xx, yy = np.meshgrid(x1grid, x2grid)

  r1, r2 = xx.flatten(), yy.flatten()
  r1, r2 = r1.reshape((len(r1), 1)), r2.reshape((len(r2), 1))

  grid = np.hstack((r1,r2))
  yhat = svc.predict(grid)
  contour_val=[xx,yy,yhat]

  return contour_val,y_pred,grid_temp


def plot_results(sample_type,classifier_type):


  if classifier_type=='MLP':
    error_matt,contour_val,y_pred=MLP(sample_type)

      #plot min error vs number of perceptrons

    tt=np.array(percept_list)
    bar1=error_matt[:,0]

    fig = plt.figure(figsize = (10, 7))
    ax2 = plt.axes()
    ax2.scatter(tt,bar1)
    ax2.plot(tt,bar1)

    plt.legend(activations)
    plt.xlabel('Number of epochs')
    plt.ylabel('Average minimum Probability of Error')
    plt.title('Minimum Error for different number of epochs')
    plt.show()

  elif classifier_type=='SVM':
    contour_val,y_pred,grid_temp=SVM(sample_type)


    # accuracy percentage SVM
```

```
    plt.plot(gamma_list,grid_temp[0:6])
    plt.plot(gamma_list,grid_temp[6:12])
    plt.plot(gamma_list,grid_temp[12:18])
    plt.plot(gamma_list,grid_temp[18:24])
    plt.plot(gamma_list,grid_temp[24:30])
    plt.plot(gamma_list,grid_temp[30:36])
    plt.xscale('log')

    plt.xlabel("Values of Gamma")
    plt.ylabel("Accuracy  in %")
    plt.legend(['C= 1000','C = 100','C = 10','C = 1.0','C = 0.1','C = 0.01'])
    plt.title("Accuracy for all combinations of C and gamma")
    plt.show()
# actual data distribution for test data
  x0=[X_test[i] for i in range(samples[1]) if Y_test[i]==0]
  x1=[X_test[i] for i in range(samples[1]) if Y_test[i]==1]

  s1=[2 for i in range(len(x0))]

  s2=[2 for i in range(len(x1))]


  plt.scatter((np.array(x0))[:,0],(np.array(x0))[:,1],s1)
  plt.scatter((np.array(x1))[:,0],(np.array(x1))[:,1],s2)

  plt.legend(['class -1','class +1'])
  plt.title('Actual Data Distribution')
  plt.show

  # contour plot of decision boundary and the classified data
  x_temp=contour_val[0]
  y_temp=contour_val[1]


  x00t = [i for i in range(10000) if (Y_test[i] == 0 and y_pred[i] == 0)]
  x01t = [i for i in range(10000) if (Y_test[i] == 0 and y_pred[i] == 1)]#fp
  x10t = [i for i in range(10000) if (Y_test[i] == 1 and y_pred[i] == 0)]#fN
  x11t = [i for i in range(10000) if (Y_test[i] == 1 and y_pred[i] == 1)]

  plt.contourf(x_temp, y_temp, contour_val[2].reshape(x_temp.shape), cmap='gist_heat')
  plt.plot(X_test[x00t,0],X_test[x00t,1],'_',color ='g', markersize = 1.95)#g
  plt.plot(X_test[x01t,0],X_test[x01t,1],'_',color = 'r', markersize = 1.95)#r
  plt.plot(X_test[x11t,0],X_test[x11t,1],'x',color ='g', markersize = 1.85)#g
  plt.plot(X_test[x10t,0],X_test[x10t,1],'x',color = 'r', markersize = 1.85)#r
```

```
    plt.legend(['class 0 correctly labeled','class 0 wrongly labeled','class 1 correctly labeled','class 1 wrongly labeled'])
    plt.xlabel("x1")
    plt.ylabel("x2")
    plt.show()

def main():
    plot_results(samples[0],'MLP')
    plot_results(samples[0],'SVM')

if __name__ == "__main__":
    activations=['relu']
    num_folds=10
    num_class=2
    features=2
    samples=[1000,10000]
    m1_n=np.zeros(2)
    m2_n=np.zeros(2)

    c1_n=np.eye(2)
    c2_n=np.eye(2)

    theta_margin=[-(np.pi),(np.pi)]
    mu_vector=[m1_n,m2_n]
    sigma_vector=[c1_n,c2_n]
    prior=np.array([0.5,0.5])
    percept_list=np.logspace(1,3,num = 15,endpoint = True,base = 5,dtype = int)
    percept_list=percept_list.tolist()

    kernel = ['rbf'] # can addd multiple kernels
    C_list = [1000, 100, 10, 1.0, 0.1, 0.01]
    gamma_list = [ 0.01,0.1, 1,10,100,1000]
    data_train,label_train=data(samples[0])
    data_test,label_test=data(samples[1])

    X_train=data_train.T
    X_test=data_test.T

    Y_train =label_train.T
    Y_test = label_test.T

    X_train = X_train.reshape(X_train.shape[0], features)
    X_test = X_test.reshape(X_test.shape[0], features)

    main()
```

Listing 1: Question 1

```
# 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
import warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)
import torchvision.transforms as transforms

from sklearn.preprocessing import minmax_scale
import cv2

import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np


def GMM():

  num_fold=10
  gmm_list=[1,2,3,4,5,6,8,10]
  data_image = cv2.imread('/content/drive/MyDrive/119082.jpg')
  feature_vector=np.zeros((data_image.shape[0] * data_image.shape[1],5))
  for i in range(data_image.shape[0]):
    for j in range(data_image.shape[1]):
      feature_vector[i*data_image.shape[1] + j,0]=i
      feature_vector[i*data_image.shape[1] + j,1]=j
      feature_vector[i*data_image.shape[1] + j,2]=data_image[i,j,2]
      feature_vector[i*data_image.shape[1] + j,3]=data_image[i,j,1]
      feature_vector[i*data_image.shape[1] + j,4]=data_image[i,j,0]

  min_max=minmax_scale(feature_vector)
  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,min_max,cv=num_fold) # this is the log likelihood score
    avg_scor=np.mean(scor)
    var_scor=np.std(scor)

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

    temp.append(avg_scor)

  model_sel=np.argmax(temp)
  model_sel=gmm_list[model_sel]
  vv=np.array(temp)


  plt.plot(gmm_list,vv,c='b', mfc='red',marker='o')
  plt.xlabel('model order')
  plt.ylabel(f'log likelihood')
  plt.title(f'log likelihood under different GMM model orders',fontsize
```

```
    =12.)
  plt.show()

  gmm_model=GaussianMixture(model_sel,covariance_type='full').fit(
    data_image.reshape((-1,3)))
  gmm_labels=gmm_model.predict(data_image.reshape((-1,3)))

  segmented=gmm_labels.reshape((data_image.shape)[0],(data_image.shape)
    [1])
  cv2.imwrite('segment.jpg',segmented)
  plt.imshow(np.array(segmented))

GMM()
```

Listing 2: Question 2