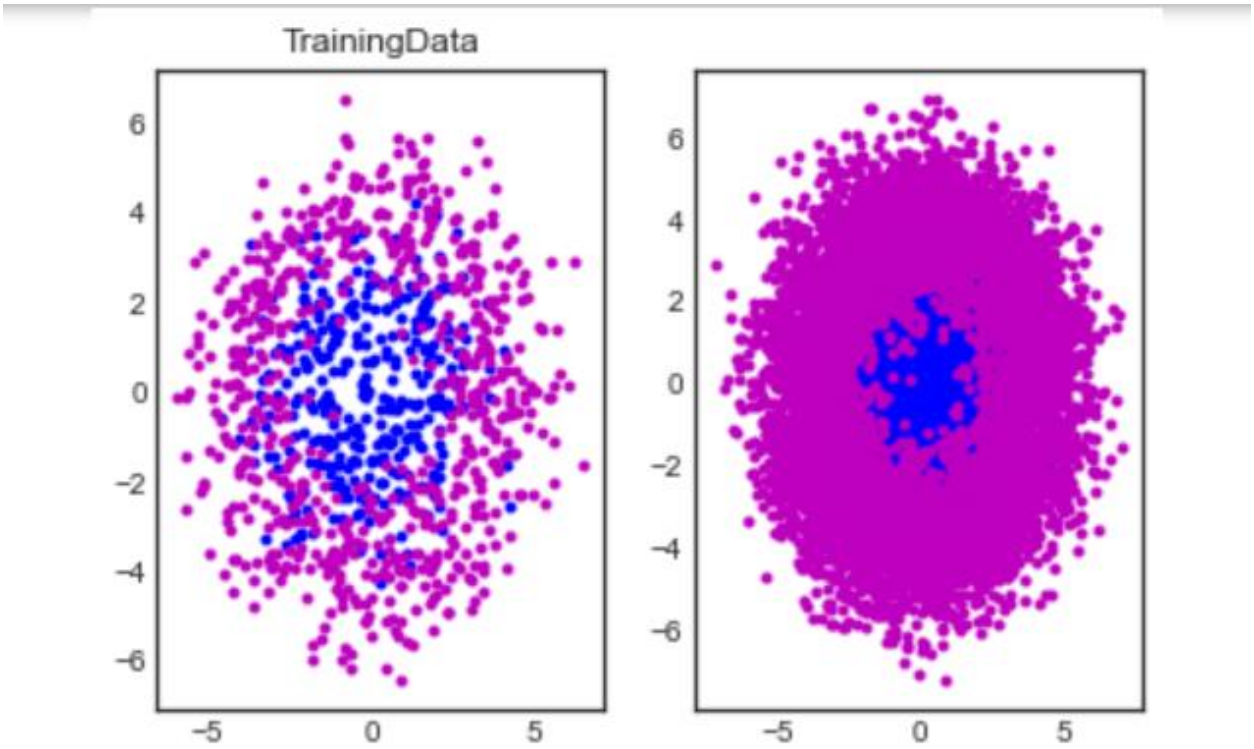
Assignment 4 : Intro to Machine Learning and Pattern Recognition

Question 1:

According to the question below are the 1000 independent and identically distributed (iid) samples for training and 10000 iid samples for testing. Here a support vector machine (SVM) classifier with a Gaussian kernel was trained to classify data generated.

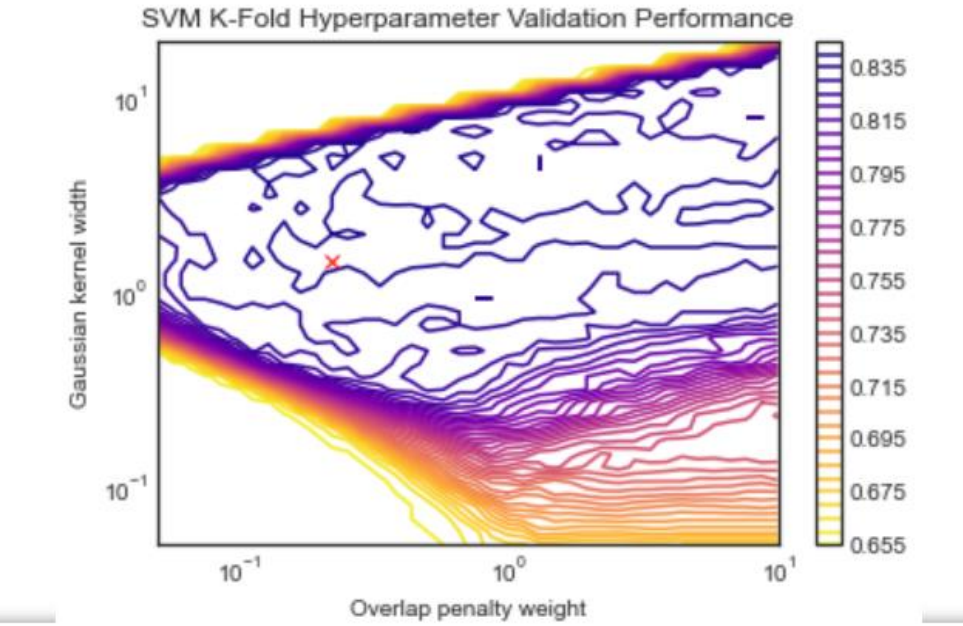
All data for class l ∈ { − 1, +1} where class C is given as $\mathbf{x} = r_l \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} + \mathbf{n}$ where θ ~Uniform[$-\pi$, π] and n ~ N(0, σ 2I). Use r − 1 = 2, r+1 = 4, σ = 1. Class priors are 0.35 for class 0 and 0.65 for class 1.
Mean class 0 = 2 , class 1= 4 , Sigma=1 .



Testing and Training data

During K-Fold validation for the SVM, 40 values of the overlap penalty weight and Gaussian kernel width were tested in the ranges [0.5, 10] and [0.5, 20], respectively. Ten-fold cross-validation was used to select the best box constraint hyperparameter C and the Gaussian kernel width hyperparameter σ. Minimum average cross-validation probability of error was used as the criteria for selecting the best hyperparameter values. Once the hyperparameters were selected the entire training set was used to train the model with the chosen hyperparameters and then the performance of this trained model was evaluated on the test dataset.

The accuracy achieved for a model with each tested combination is shown in the contour plot below.

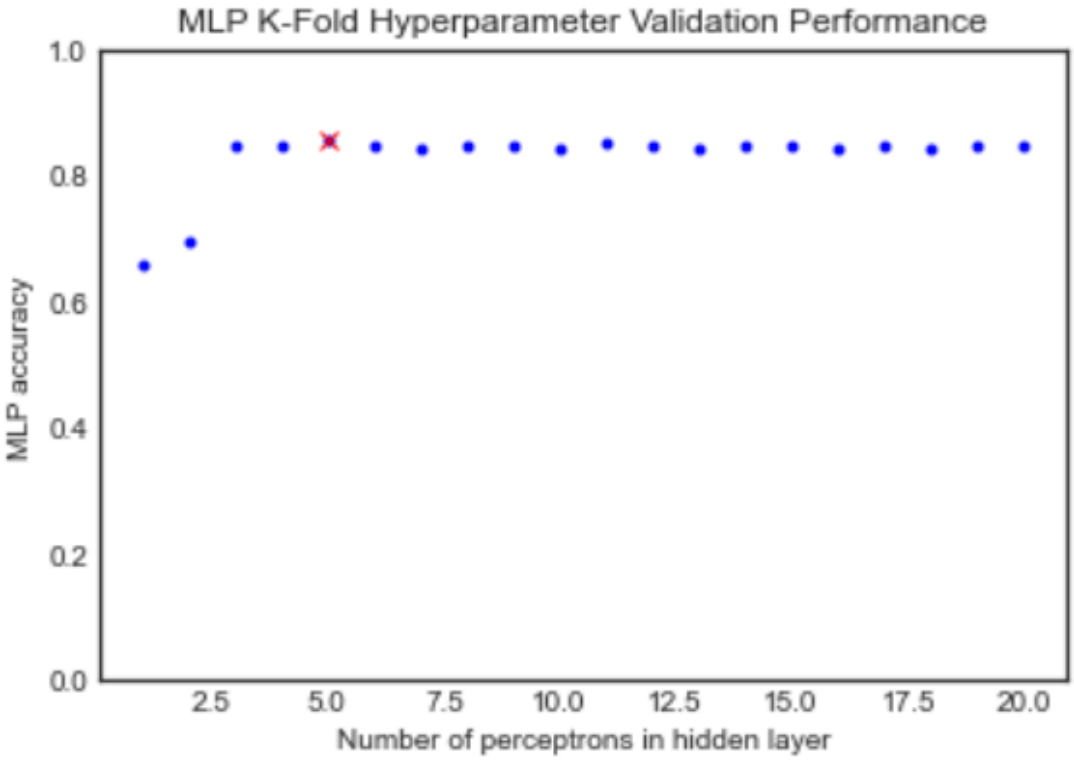


SVM K-Fold Hyperparameter Validation Performance

The maximum achieved accuracy occurred using an overlap penalty weight of 0.22283246506576698 and a Gaussian kernel width of 1.47, as marked with a red 'x' on the plot above. **The best svm accuracy while training is is 0.845**. This combination produced an average accuracy of 0.845 on the 10 K-Fold validation partitions. The plot above increasing flat plateaus, both on the lower and upper bounds of accuracy. With a low enough overlap penalty weight and kernel width, samples are essentially "too far" from each other to produce meaningfully clustered groups. With a low overlap penalty weight and large kernel width, samples become "too close" to each other and cannot be meaningfully separated. Given the right balance between these two parameters, it is possible to derive an appropriate decision boundary, but this boundary cannot do anything to correctly classify samples in the overlapping Gaussian distributions.

. The classification boundary appears roughly circular, as is to be expected according to the method of data generation.

During K-Fold validation for the MLP model, up to 5 perceptron were tested in the hidden layer. The accuracy achieved for each model is shown in the plot below.

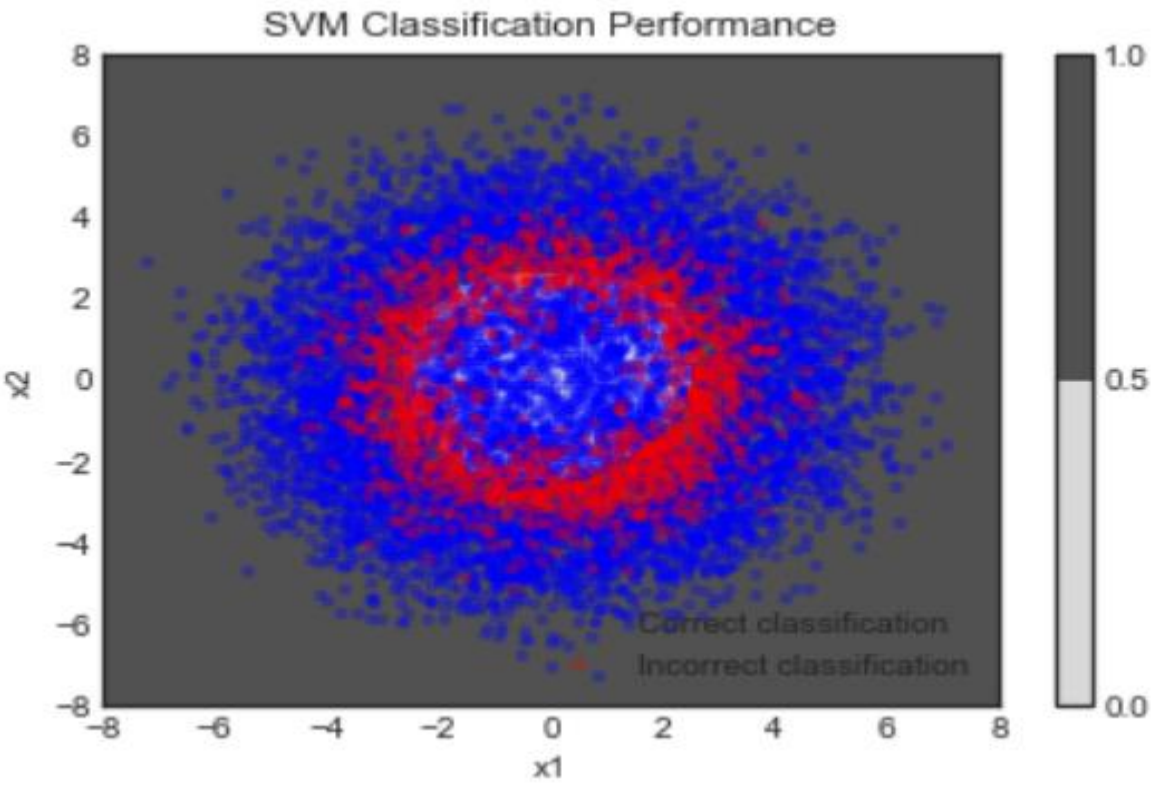


The best MLP accuracy(Kfold hyperparameter) was achieved with 5 perceptron

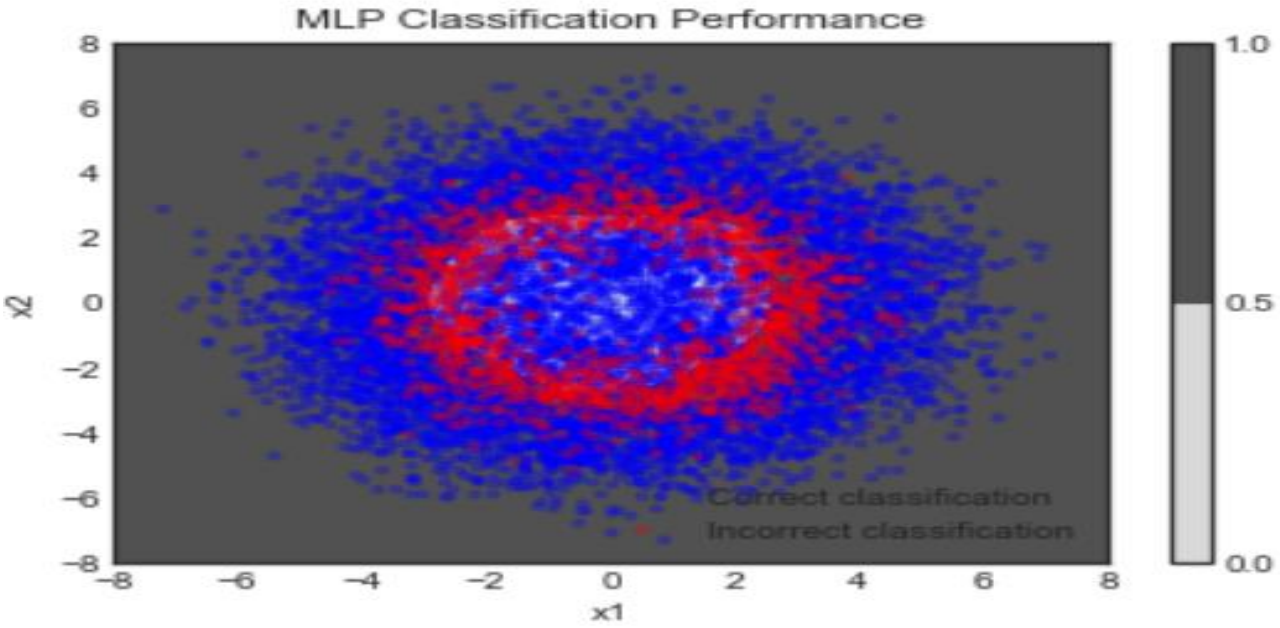
The maximum achieved accuracy occurred using 5 perceptron in the hidden layer, as marked with a red 'x' on the plot above. **The best mlp accuracy while training is 0.855.** This configuration produced an average accuracy of 0.845 on the 10 K-Fold validation partitions. After the 5 parameter the perceptron tends to remain constant generally hence we consider the accuracy is achieved with the help of 5 perceptron.

Similarly to the SVM results, there is a fairly smooth plateau that occurs at the maximum accuracy for the data in question. The optimal model selected by K-Fold validation had 5 perceptron, but other quantities greater than 2 perform about the same as well.

The below figure was generated by training the optimal SVM model selected by K-Fold validation on the entire test dataset. The red color denotes the incorrect classification. The blue color denotes the right classification



The below figure was generated by training the optimal MLP model selected by K-Fold validation on the entire test dataset.



The model above was fit with an accuracy of 0.84090. Once again, the classification boundary appears roughly circular. **The test dataset was fit by the SVM model with an accuracy of 0.8382. The test dataset was fit by the MLP model with an accuracy of 0.8409000039100647.**
The MLP classifier was able to achieve slightly better accuracy than the SVM classifier, although it takes much longer to train. Visually, the smoother boundary generated by the MLP model is closer to the ideal, circular case than the more jagged boundary created by the SVM model. However, both models already perform extremely close to maximum accuracy, which would be at approximately 85% accuracy (**or a 15% probability of error).**

Question 2:

Below is **the Figure(1)** of the original image GMM-based clustering and its 2-Component image segmented by taking the color of image into consideration . K mean Clustering is a type of the unsupervised learning which is used when you have unlabeled data. The goal is to find groups in the data with the number of groups represented by the variable K.

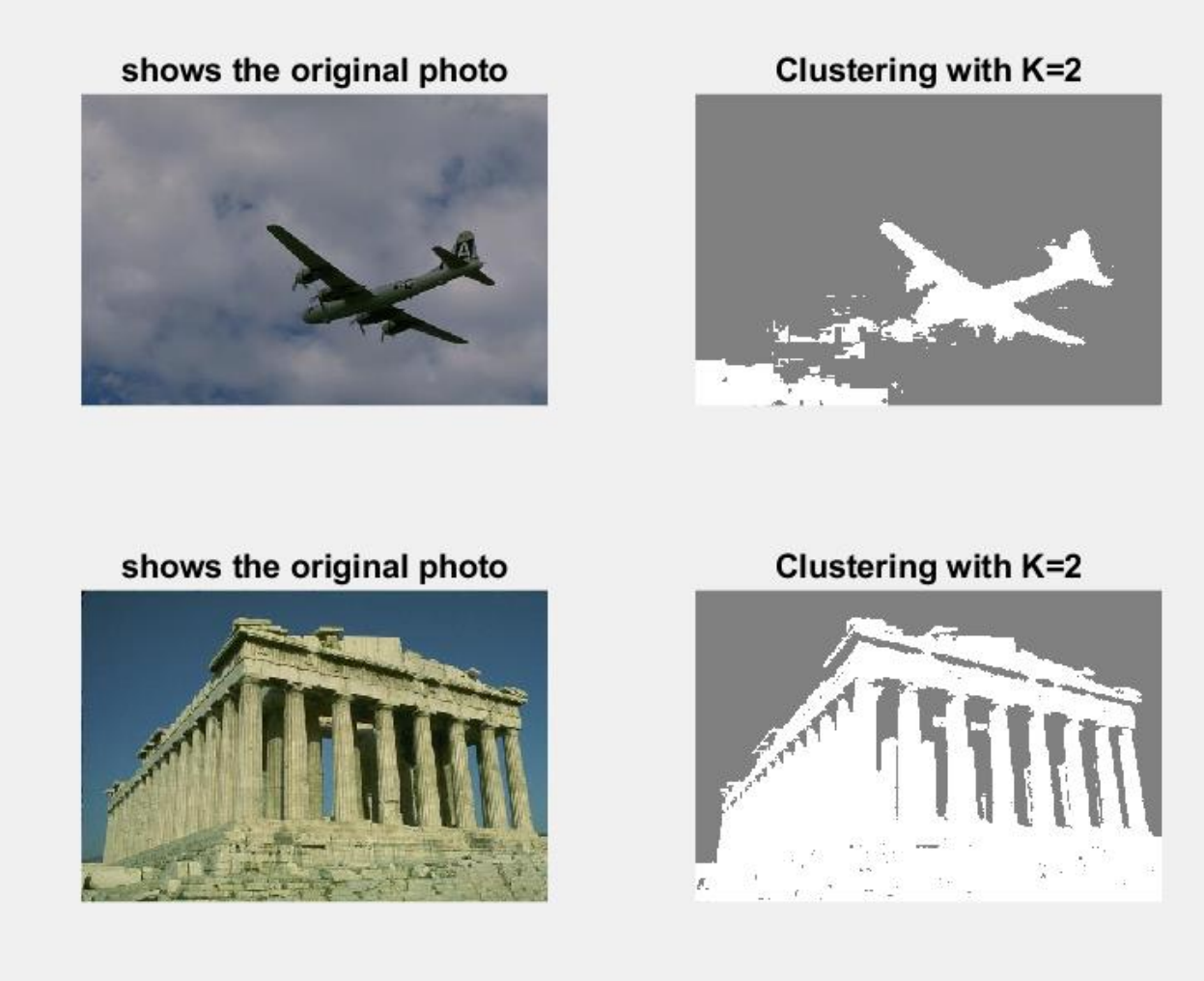


Figure 1

GMM-based clustering was used to segment 2 different images- an airplane image and a building image.

First to take out the data from the images had to be normalized across all the dimensions.

A 5-dimensional feature vector was generated for each image, with the dimensions being row index, column index, and red, green, and blue values. Each individual feature was mapped into a [0,1] range in this pre-processing stage. This was done for all pixels of each image separately, giving a 5- dimensional (data) hypercube of size 154401.

Once the normalized feature vector was generated, each image was first fit to a 2-component GM Model using the built in fitgmdist function in Matlab. Following that, the cluster function was used to assign labels to each pixel of the image. After reshaping the image into its original size, the 2-component image was displayed as shown above (K=2)

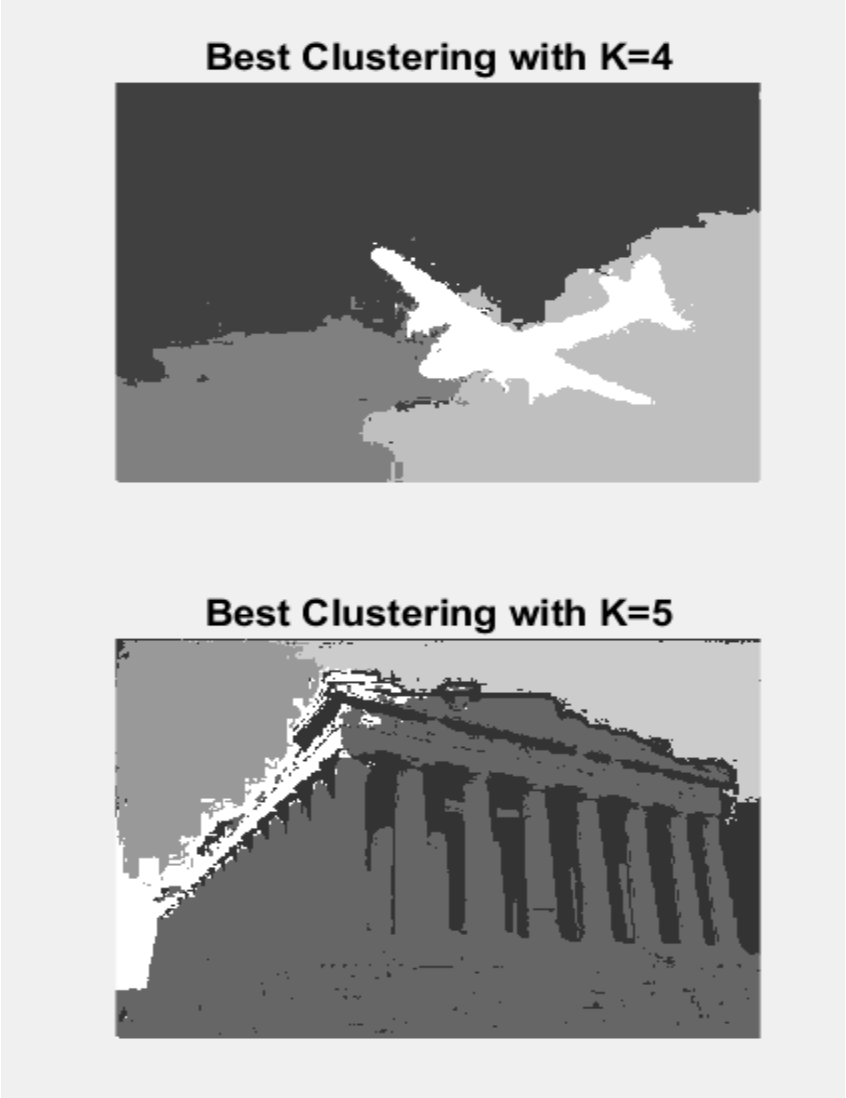


Figure 2

The above figure **(Figure2)** is the final best fit GMM model . In this the image was fit and the cluster function was used again to label each pixel with the cluster number. Here the best fit for the air plane is 4 and for the monument is 5 . This number describes that each different from the other in terms of the properties that has been detected while training . To fit the GMM we have used maximum likelihood parameter estimation and 10-fold cross validation (with maximum average validation-log-likelihood as the objective) for model order selection. Once the best GMM was found the feature vectors, we assigned the most likely component label to each pixel by evaluating component label posterior probabilities for each feature vector according to GMM.

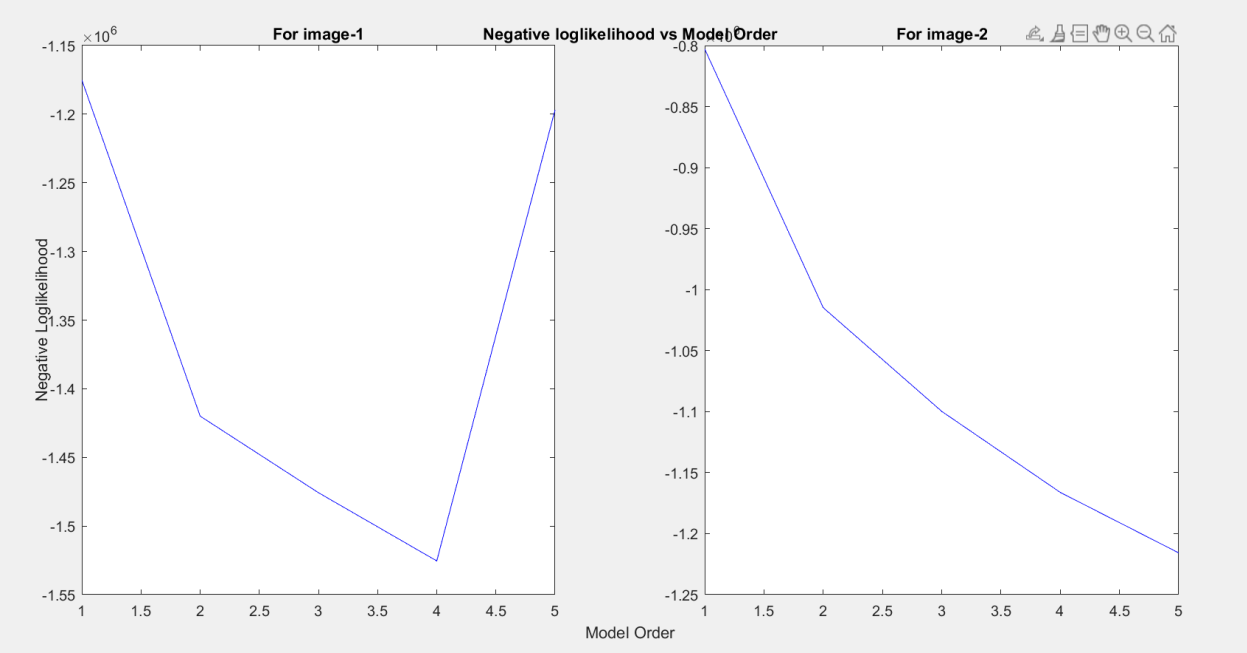


Figure 3

Figure 3 shows the max log likelihood function and model order relation.

For each image, 10-fold cross-validation was used to fit the image to various model orders (from 1to 5), and the average log-likelihood was used to establish which model order is the best fit for each image. The hypercube data was fit to a GMM model and with the objective function of average- max log-likelihood obtained from 10-fold cross validation, the best number of clusters or classes were chosen. As can be observed, for both images, the algorithm picked 4 clusters as the best fit for airplane and 5 for the monument. This shows the average log likelihood for each model order and for each image. Each column number represents the model order, and as expected, the last column (model order 5), has the greatest log likelihood of the component values tested.

Appendix

Programs of Question 1 :

Python

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import StratifiedKFold
from sklearn.svm import SVC
import keras
from keras.models import Sequential
from keras.layers import Dense
from tensorflow.keras.optimizers import SGD
plotData = True
```

```
n = 2
Ntrain = 1000
Ntest = 10000
ClassPriors = [0.35, 0.65]
r0 = 2
r1 = 4
sigma = 1
def generate_data(N):
   data_labels = np.random.choice(2, N, replace=True, p=ClassPriors)
   ind0 = np.array((data_labels==0).nonzero())
   ind1 = np.array((data_labels==1).nonzero())
   N0 = np.shape(ind0)[1]
   N1 = np.shape(ind1)[1]
   theta0 = 2*np.pi*np.random.standard_normal(N0)
   theta1 = 2*np.pi*np.random.standard_normal(N1)
   x0 = sigma**2*np.random.standard_normal((N0,n)) + r0 * np.transpose([np.cos(theta0),
np.sin(theta0)])
   x1 = sigma**2*np.random.standard_normal((N1,n)) + r1 * np.transpose([np.cos(theta1),
np.sin(theta1)])
   data_features = np.zeros((N, 2))
   np.put_along_axis(data_features, np.transpose(ind0), x0, axis=0)
   np.put_along_axis(data_features, np.transpose(ind1), x1, axis=0)
   return (data_labels, data_features)
def plot_data(TrainingData_labels, TrainingData_features, TestingData_labels, TestingData_features):
   plt.subplot(1,2,1)
   plt.plot(TrainingData_features[np.array((TrainingData_labels==0).nonzero())][0,:,0],
         TrainingData_features[np.array((TrainingData_labels==0).nonzero())][0,:,1],
         'b.')
   plt.plot(TrainingData_features[np.array((TrainingData_labels==1).nonzero())][0,:,0],
         TrainingData_features[np.array((TrainingData_labels==1).nonzero())][0,:,1],
         'm.')
   plt.title('TrainingData')

   plt.subplot(1,2,2)

   plt.plot(TestingData_features[np.array((TestingData_labels==0).nonzero())][0,:,0],
         TestingData_features[np.array((TestingData_labels==0).nonzero())][0,:,1],
         'b.')
   plt.plot(TestingData_features[np.array((TestingData_labels==1).nonzero())][0,:,0],
         TestingData_features[np.array((TestingData_labels==1).nonzero())][0,:,1],
         'm.')
   plt.show()
# Uses K-Fold cross validation to find the best hyperparameters for an SVM model, and plots the results
def train_SVM_hyperparams(TrainingData_labels, TrainingData_features):
```

```
  hyperparam_candidates = np.meshgrid(np.geomspace(0.05, 10, 40), np.geomspace(0.05, 20, 40))
  hyperparam_performance = np.zeros((np.shape(hyperparam_candidates)[1] *
np.shape(hyperparam_candidates)[2]))
  for (i, hyperparams) in enumerate(np.reshape(np.transpose(hyperparam_candidates), (-1, 2))):
    skf = StratifiedKFold(n_splits=K, shuffle=False)

    total_accuracy = 0

    for(k, (train, test)) in enumerate(skf.split(TrainingData_features, TrainingData_labels)):
      (_, accuracy) = SVM_accuracy(hyperparams, TrainingData_features[train],
TrainingData_labels[train], TrainingData_features[test], TrainingData_labels[test])
      total_accuracy += accuracy

    accuracy = total_accuracy / K
    hyperparam_performance[i] = accuracy

    print(i, accuracy)

  plt.style.use('seaborn-white')
  ax = plt.gca()
  ax.set_xscale('log')
  ax.set_yscale('log')

  max_perf_index = np.argmax(hyperparam_performance)
  max_perf_x1 = max_perf_index % 40
  max_perf_x2 = max_perf_index // 40
  best_overlap_penalty = hyperparam_candidates[0][max_perf_x1][max_perf_x2]
  best_kernel_width = hyperparam_candidates[1][max_perf_x1][max_perf_x2]

  plt.contour(hyperparam_candidates[0], hyperparam_candidates[1],
np.transpose(np.reshape(hyperparam_performance, (40, 40))), cmap='plasma_r', levels=40);
  plt.title("SVM K-Fold Hyperparameter Validation Performance")
  plt.xlabel("Overlap penalty weight")
  plt.ylabel("Gaussian kernel width")
  plt.plot(best_overlap_penalty, best_kernel_width, 'rx')
  plt.colorbar()
  print("The best SVM accuracy was " + str(hyperparam_performance[max_perf_index]) + ".")
  plt.show()
  return (best_overlap_penalty, best_kernel_width)
# Trains an SVM with the given hyperparameters on the train data, then validates its performance on
the given test data.
# Returns the trained model and respective validation loss.
def SVM_accuracy(hyperparams, train_features, train_labels, test_features, test_labels):
  (overlap_penalty, kernel_width) = hyperparams
```

```
    model = SVC(C=overlap_penalty, kernel='rbf', gamma=1/(2*kernel_width**2))
    model.fit(train_features, train_labels)
    predictions = model.predict(test_features)
    num_correct = len(np.squeeze((predictions == test_labels).nonzero()))
    accuracy = num_correct / len(test_features)
    return (model, accuracy)
# Uses K-Fold cross validation to find the best hyperparameters for an MLP model, and plots the results
def train_MLP_hyperparams(TrainingData_labels, TrainingData_features):
    hyperparam_candidates = list(range(1, 21))
    hyperparam_performance = np.zeros(np.shape(hyperparam_candidates))
    for (i, hyperparams) in enumerate(hyperparam_candidates):
        skf = StratifiedKFold(n_splits=K, shuffle=False)

        total_accuracy = 0

        for(k, (train, test)) in enumerate(skf.split(TrainingData_features, TrainingData_labels)):
            accuracy = max(map(lambda _: MLP_accuracy(hyperparams, TrainingData_features[train],
TrainingData_labels[train], TrainingData_features[test], TrainingData_labels[test])[1], range(4)))
            total_accuracy += accuracy

        accuracy = total_accuracy / K
        hyperparam_performance[i] = accuracy

        print(i, accuracy)

    plt.style.use('seaborn-white')

    max_perf_index = np.argmax(hyperparam_performance)
    best_num_perceptrons = hyperparam_candidates[max_perf_index]

    plt.plot(hyperparam_candidates, hyperparam_performance, 'b.')
    plt.title("MLP K-Fold Hyperparameter Validation Performance")
    plt.xlabel("Number of perceptrons in hidden layer")
    plt.ylabel("MLP accuracy")
    plt.ylim([0,1])
    plt.plot(hyperparam_candidates[max_perf_index], hyperparam_performance[max_perf_index], 'rx')
    print("The best MLP accuracy was " + str(hyperparam_performance[max_perf_index]) + ".")
    plt.show()
    return best_num_perceptrons
# Trains an MLP with the given number of perceptrons on the train data, then validates its performance
on the given test data.
# Returns the trained model and respective validation loss.
def MLP_accuracy(num_perceptrons, train_features, train_labels, test_features, test_labels):
    sgd = SGD(lr=0.05, momentum=0.9)
```

```
    model = Sequential()
    model.add(Dense(num_perceptrons, activation='sigmoid', input_dim=2))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])
    model.fit(train_features, train_labels, epochs=300, batch_size=100, verbose=0)
    (loss, accuracy) = model.evaluate(test_features, test_labels)
    return (model, accuracy)
# Creates a contour plot for a model, along with colored samples based on their classifications by the
model.
def plot_trained_model(model_type, model, features, labels):
    predictions = np.squeeze(model.predict(features))
    correct = np.array(np.squeeze((np.round(predictions) == labels).nonzero()))
    incorrect = np.array(np.squeeze((np.round(predictions) != labels).nonzero()))

    plt.plot(features[correct][:,0],
            features[correct][:,1],
            'b.', alpha=0.25)
    plt.plot(features[incorrect][:,0],
            features[incorrect][:,1],
            'r.', alpha=0.25)
    plt.title(model_type + ' Classification Performance')
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.legend(['Correct classification', 'Incorrect classification'])

    gridpoints = np.meshgrid(np.linspace(-8, 8, 128), np.linspace(-8, 8, 128))
    contour_values = np.transpose(np.reshape(model.predict(np.reshape(np.transpose(gridpoints), (-1,
2))), (128, 128)))
    plt.contourf(gridpoints[0], gridpoints[1], contour_values, levels=1);
    plt.colorbar();

    plt.show()

K = 10


(TrainingData_labels, TrainingData_features) = generate_data(Ntrain)
(TestingData_labels, TestingData_features) = generate_data(Ntest)

if plotData:
    plot_data(TrainingData_labels, TrainingData_features, TestingData_labels, TestingData_features)

SVM_hyperparams = train_SVM_hyperparams(TrainingData_labels, TrainingData_features)
MLP_hyperparams = train_MLP_hyperparams(TrainingData_labels, TrainingData_features)
```

```
(overlap_penalty, kernel_width) = SVM_hyperparams
print("The best SVM accuracy was achieved with an overlap penalty weight of " + str(overlap_penalty) +
" and a Gaussian kernel width of " + str(kernel_width) + ".")
print("The best MLP accuracy was achieved with " + str(MLP_hyperparams) + " perceptrons.")

(SVM_model, SVM_performance) = SVM_accuracy(SVM_hyperparams, TrainingData_features,
TrainingData_labels, TestingData_features, TestingData_labels)
(MLP_model, MLP_performance) = max(map(lambda _: MLP_accuracy(MLP_hyperparams,
TrainingData_features, TrainingData_labels, TestingData_features, TestingData_labels), range(5)),
key=lambda r: r[1])

print("The test dataset was fit by the SVM model with an accuracy of " + str(SVM_performance) + ".")
print("The test dataset was fit by the MLP model with an accuracy of " + str(MLP_performance) + ".")

plot_trained_model('SVM', SVM_model, TestingData_features, TestingData_labels)
plot_trained_model('MLP', MLP_model, TestingData_features, TestingData_labels)
```

Question 2 code : MATLAB

```
%% Initialize
clear all; close all;
f{1,1} = "3096_color.jpg";
f{1,2} = "67079.jpg";
K = 10;
M = 5;
n = size(f, 2);
%%
for i = 1:n
    imdata  = imread(f{1,i});
    figure(1), subplot(n, 2, i*2-1),
    imshow(imdata);
    title("shows the original photo"); hold on;
    [R,C,D] = size(imdata); N = R*C; imdata = double(imdata);
    rowIndices = (1:R)'*ones(1,C); colIndices = ones(R,1)*(1:C);
    features = [rowIndices(:)';colIndices(:)']; % initialize with row and column
indices
    for d = 1:D
        imdatad = imdata(:,:,d); % pick one color at a time
        features = [features;imdatad(:)'];
    end
    minf = min(features,[],2); maxf = max(features,[],2);
    ranges = maxf-minf;
    x = diag(ranges.^(-1))*(features-repmat(minf,1,N));
    d = size(x,1);
    model = 2;
    gm = fitgmdist(x',model);
    p = posterior(gm, x');
    [~, l] = max(p,[], 2);
    li = reshape(l, R, C);
    figure(1), subplot(n, 2, i*2)
    imshow(uint8(li*255/model));
    title(strcat("Clustering with K=", num2str(model)));
    ab = zeros(1,M);
```

```
    for model = 1:M
        ab(1,model) = calcLikelihood(x, model, K);
    end
    [~, mini] = min(ab);
    gm = fitgmdist(x', mini);
    p = posterior(gm, x');
    [~, l] = max(p,[], 2);
    li = reshape(l, R, C);
    figure(2), subplot(n,1,i),
    imshow(uint8(li*255/mini));
    title(strcat("Best Clustering with K=", num2str(mini)));
    fig = figure(3);
    subplot(1,n,i), plot(ab,'-b');
    title(strcat("For image-", num2str(i)));
end
han = axes(fig, 'visible', 'off');
han.Title.Visible='on';
han.XLabel.Visible='on';
han.YLabel.Visible='on';
ylabel(han,'Negative Loglikelihood');
xlabel(han,'Model Order');
title(han,'Negative loglikelihood vs Model Order');
%% function
function negativeLoglikelihood = calcLikelihood(x, model, K)
    N = size(x,2);
    dummy = ceil(linspace(0, N, K+1));
    negativeLoglikelihood = 0;
    for k=1:K
        indPartitionLimits(k,:) = [dummy(k) + 1, dummy(k+1)];
    end
    for k = 1:K
        indValidate = (indPartitionLimits(k,1):indPartitionLimits(k,2));
        xv = x(:, indValidate); % Using folk k as validation set
        if k == 1
            indTrain = (indPartitionLimits(k,2)+1:N);
        elseif k == K
            indTrain = (1:indPartitionLimits(k,1)-1);
        else
            indTrain = (indPartitionLimits(k-1,2)+1:indPartitionLimits(k+1,1)-1);
        end
        xt = x(:, indTrain);
        try
            gm = fitgmdist(xt', model);
            [~, nlogl] = posterior(gm, xv');
            negativeLoglikelihood = negativeLoglikelihood + nlogl;
        catch exception
        end
    end
end
```