EECE5644 Fall 2021 – Take Home Exam 4

Balaji Sundareshan

Question 1

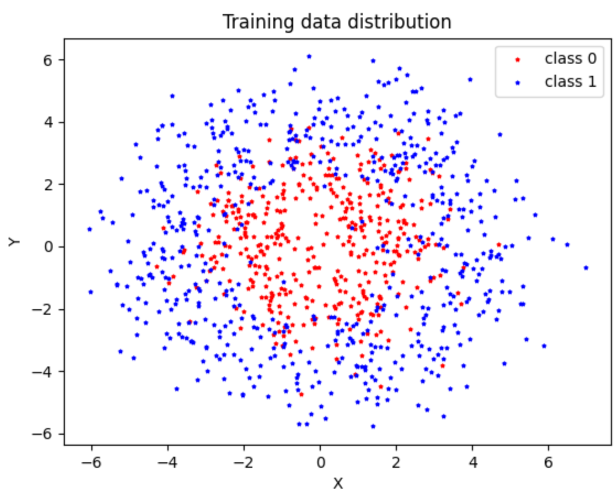
For this question, 1000 independent and identically distributed (iid) samples were generated for training and 10000 iid samples for testing. The data was generated as per the below parameters

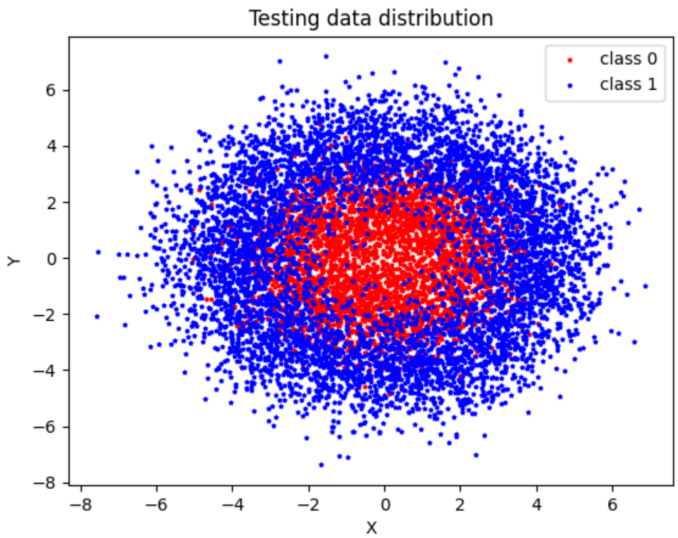
$$\mathbf{x} = r_l \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} + \mathbf{n}$$

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

The class priors were 0.4 and 0.6 for class 0 and class 1 respectively.

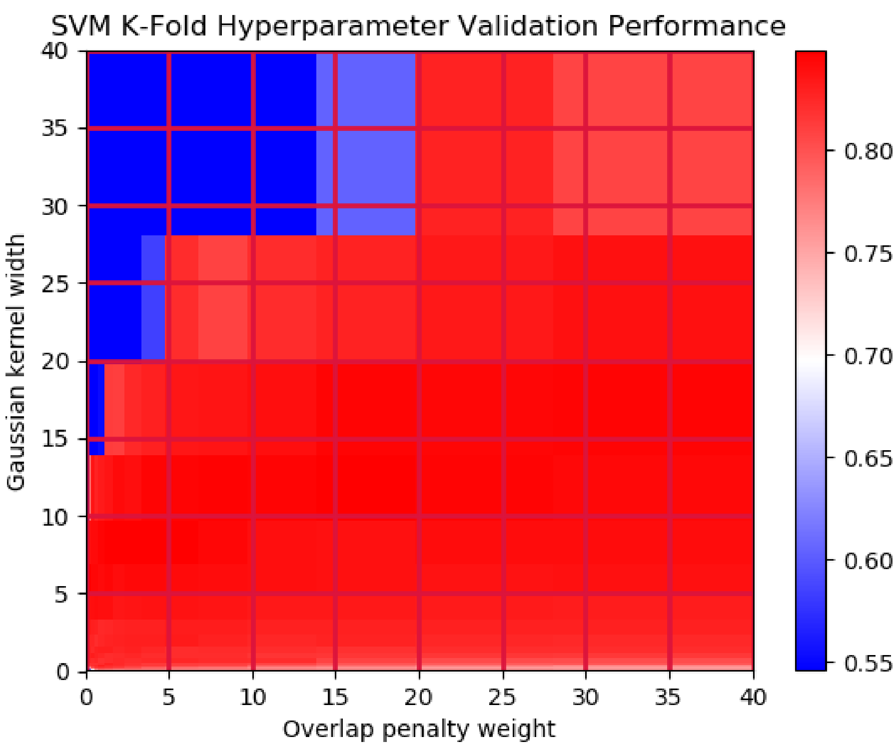
Plots generated for the above mentioned train and test datasets are shown below.





SVM Classifier:

During K-Fold validation for the SVM, 20 values of the box constraint parameter(C) and Gaussian kernel width(K) were evaluated in the ranges [0.5, 40] and [0.5, 40], respectively. The accuracy achieved for a model with each tested combination is shown in the below heatmap



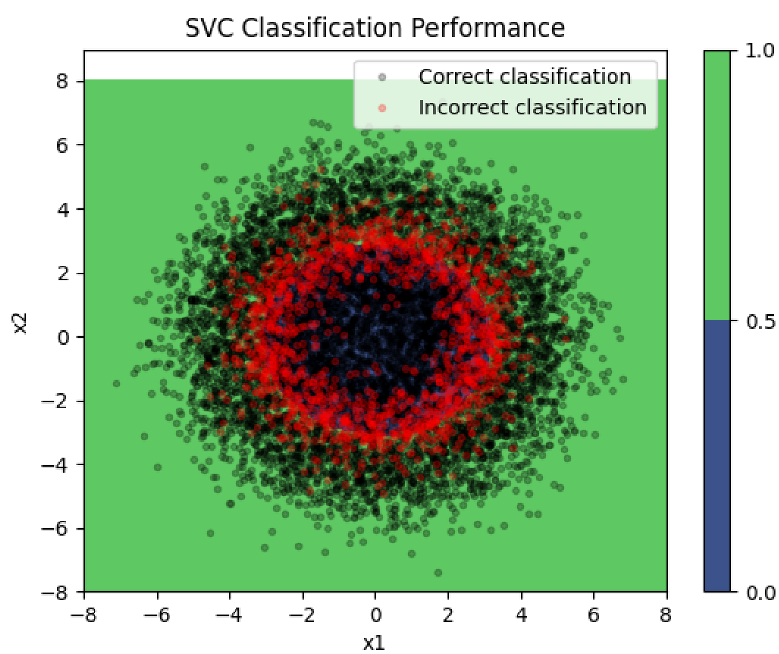
The maximum accuracy (**0.85**) was achieved on the training dataset using 10-fold cross-validation when hyperparameters C is **1.186** and K is **6.88**. Gamma was calculated from gaussian kernel width as

Gamma = 1/(2 * kernel width**2)

The optimal gamma value is **0.01.** With these hyperparameters, the Support Vector Classifier was trained on the entire training dataset and validated on the test dataset containing 10,000 samples. The accuracy on the test dataset was **0.828.**

Below plot shows the classification results of the Support Vector Classifier trained on 1000 samples and tested on 10,000 samples. As the data distribution of both the classes are circular, the decision boundary separating these two classes are also circular. The decision boundary was calculated using the confidence threshold as **0.5** and was superimposed in the below plot.

Black dots and red dots represent correct classification and incorrect classification respectively. All the samples classified as class 0 and class 1 are marked in blue and green respectively. As there is some overlap between two classes, there are more misclassifications in the decision boundary as expected.



MLP:

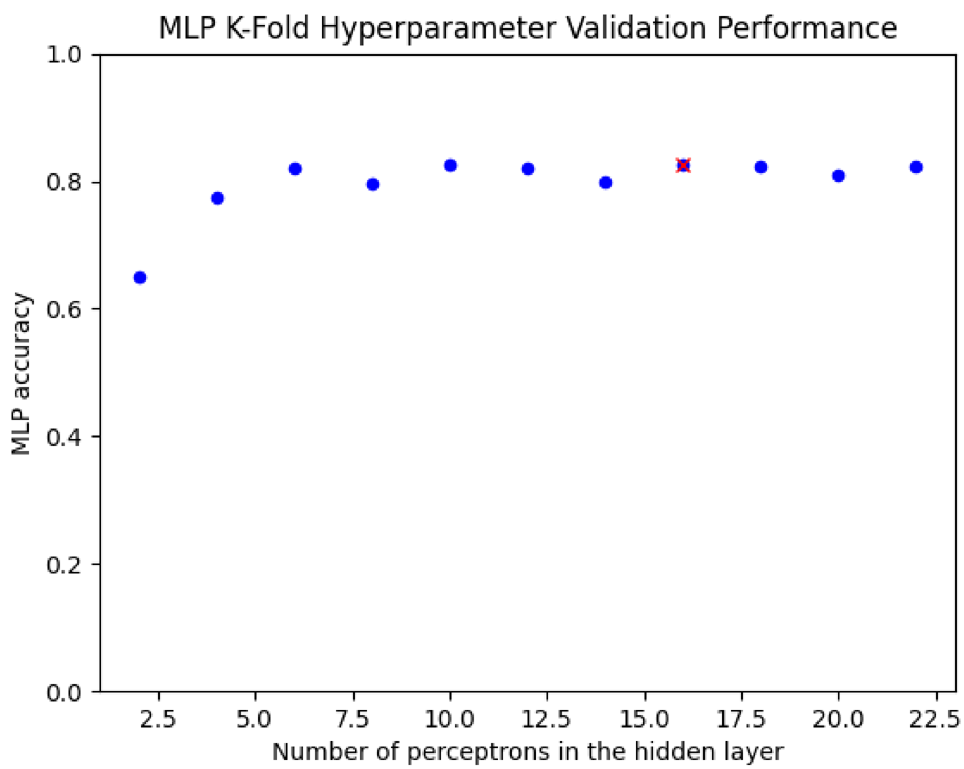
For this part, I implemented a 2-layer MLP with one hidden layer and one output layer. RELU was used as the activation function for the hidden layer. As there are only two classes, sigmoid was used as the activation function for the output layer.

The loss function used was binary cross-entropy loss and the optimizer used was Stochastic Gradient descent. The model was trained for 300 epochs.

The optimal number of perceptrons for the hidden layer was estimated by calculating accuracy achieved on 10-fold cross-validation for a range of values. For this part, the range of values used for finding an optimal number of perceptrons was from 2 to 22.

The optimal number of perceptrons for which maximum accuracy was achieved on the training dataset using 10-fold cross-validation was **16.** The accuracy achieved on the training dataset was **0.825**.

Below plot shows the accuracy of the MLP model for a varying number of perceptrons in the hidden layer.

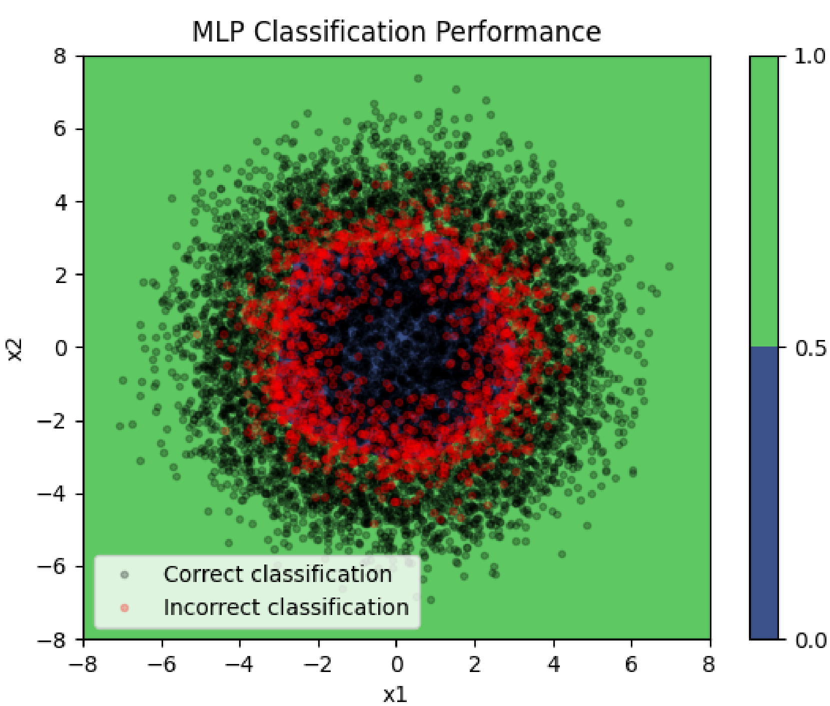


Below snapshot shows the architecture of the final MLP with optimal number of perceptrons used to train on the entire training dataset. This final model was used to validate on the test dataset. The accuracy on the test dataset is **0.838.**

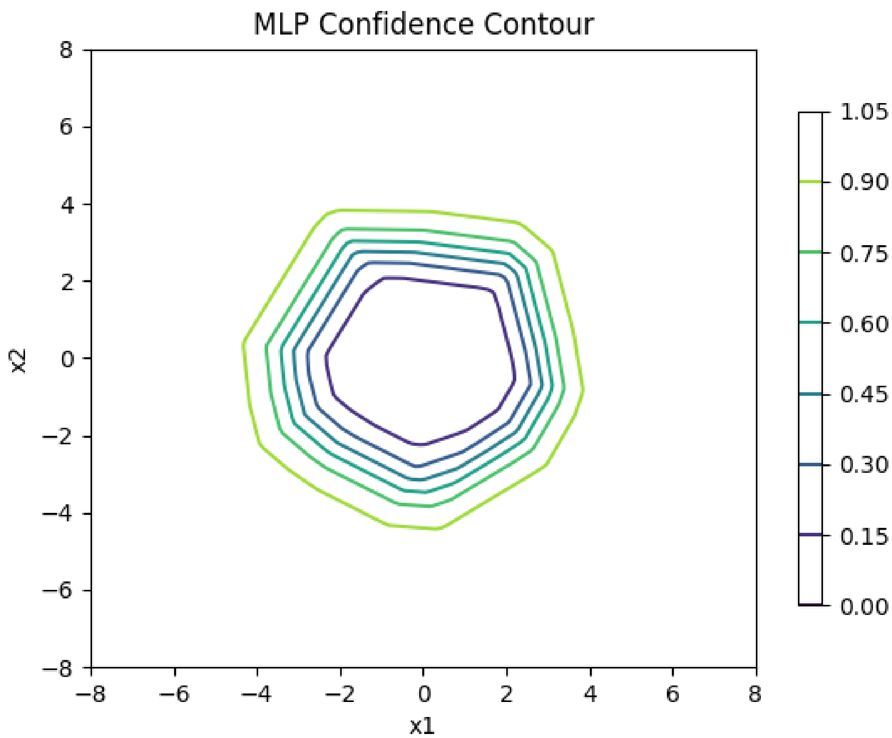
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (100, 16)                 48

 dense_1 (Dense)             (100, 1)                  17

=================================================================
Total params: 65
Trainable params: 65
Non-trainable params: 0
_________________________________________________________________
```

Below plot shows the classification results of the MLP Classifier trained on 1000 samples and tested on 10,000 samples. The threshold used to classify both the classes was **0.5.**

Black dots and red dots represent correct classification and incorrect classification respectively. All the samples classified as class 0 and class 1 are marked in blue and green respectively. As there is some overlap between two classes, there are more misclassifications in the decision boundary as expected.



Below plot shows the confidence contour of the MLP model on the test dataset for different confidence values.



Model	Train Accuracy(10-fold) 1000 samples	Test Accuracy 10000 samples
SVM	0.85	0.828
MLP	0.825	0.84

The training accuracy is higher for SVM compared to the MLP classifier. But the classification accuracy on the test dataset for MLP is higher compared to the SVM classifier.

As the number of data samples in the test dataset is high, there are many samples in the boundary between two classes. As the SVM classifier is sensitive to the data samples near the boundary, this increase in data samples caused reduction in accuracy when validating on the test dataset.

This is not the case for the MLP classifier. The accuracy increased on the test dataset.

Question 2

For this question, a 5-dimensional feature vector was calculated from the input image using row index, column index, red, green and blue pixel values. Then a Gaussian Mixture Model was fit using maximum likelihood parameter estimation. The optimal number of Gaussian components was estimated by using maximum average validation-log-likelihood as the objective function and 10-fold cross validation.

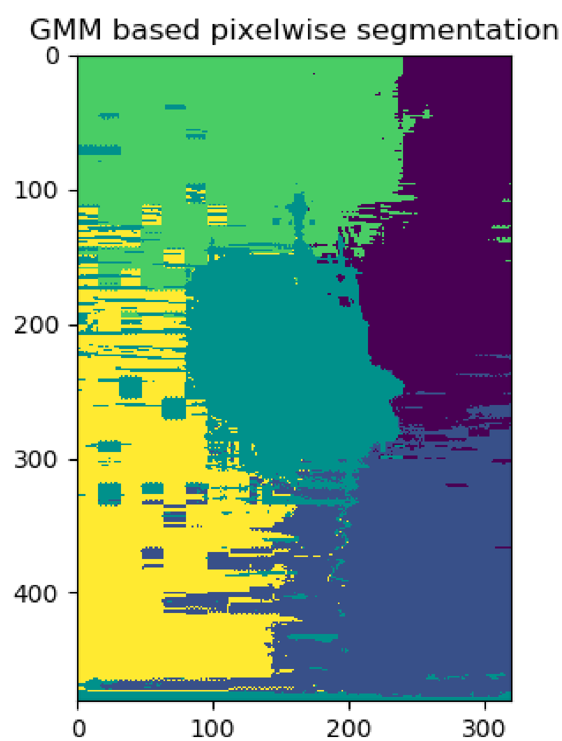
All the RGB images used here are of dimension (321, 481). So the number of pixels are 321 X 481 pixels = 1,54,401 pixels which each pixel contains 3 values for R, G and B.

Each of the 5 features was normalized to [0,1] scale using the normalize function from sklearn library. GMM was fitted using GaussianMixture() function from scipy library.

The pdf was calculated for the normalized feature vector for each cluster and for each pixel, the cluster with maximum probability was mapped to that particular pixel.

Following are three RGB images and their GMM based pixel wise segmentation. For all the below images, the optimal number of Gaussian components was estimated by calculating maximum average validation-log-likelihood for a range of values varying from 1 to 11.

Image 1:



Below plot shows the average validation-log-likelihood for a varying number of gaussian components. For this image, the optimal number of gaussian components is **5.**(marked in red)

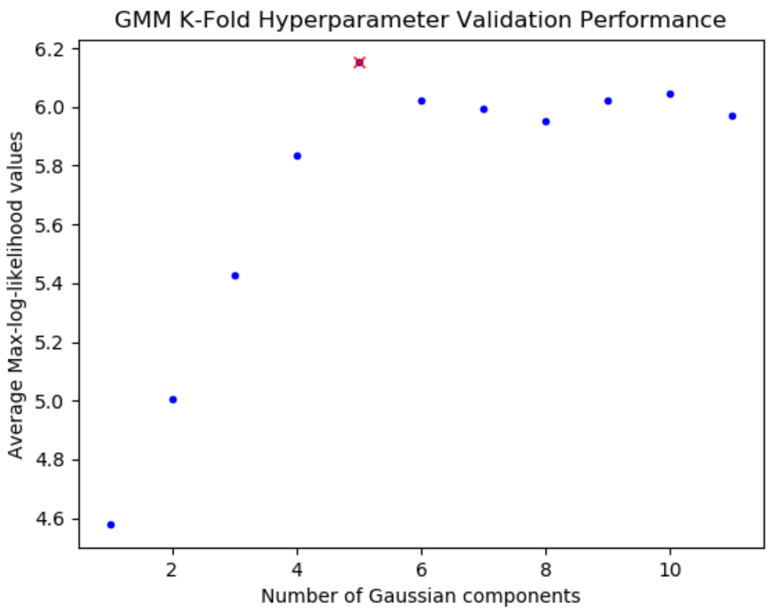
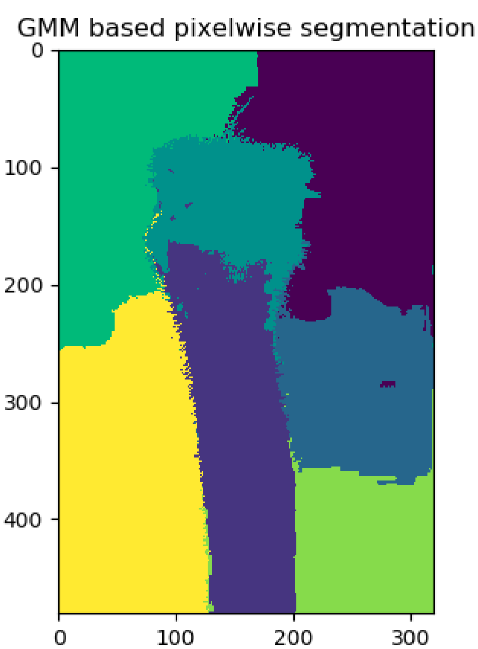


Image 2:



Below plot shows the average validation-log-likelihood for a varying number of gaussian components. For this image, the optimal number of gaussian components is **7.**(marked in red)

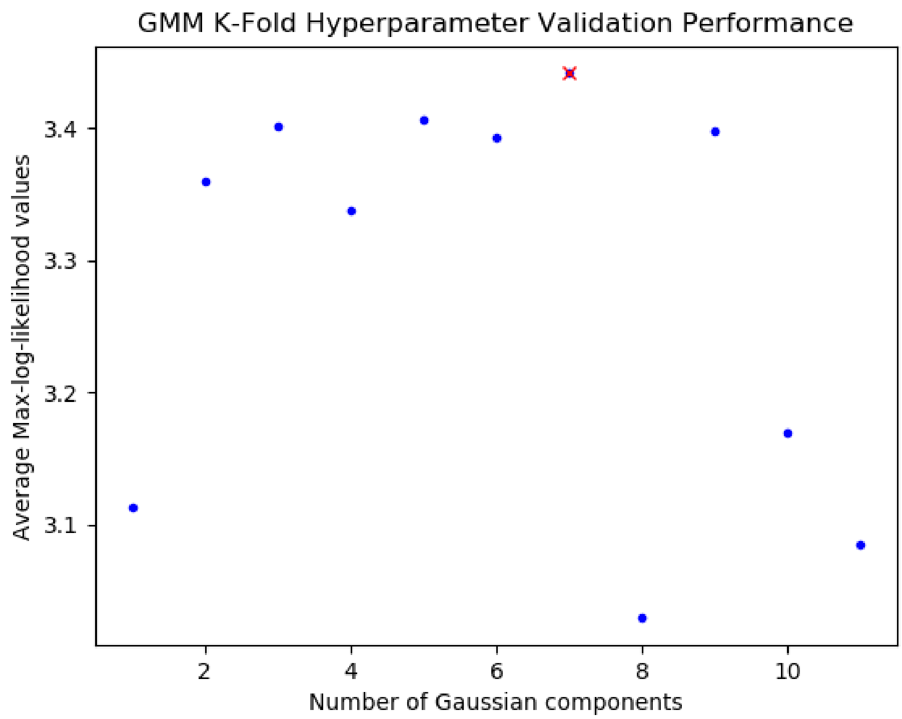
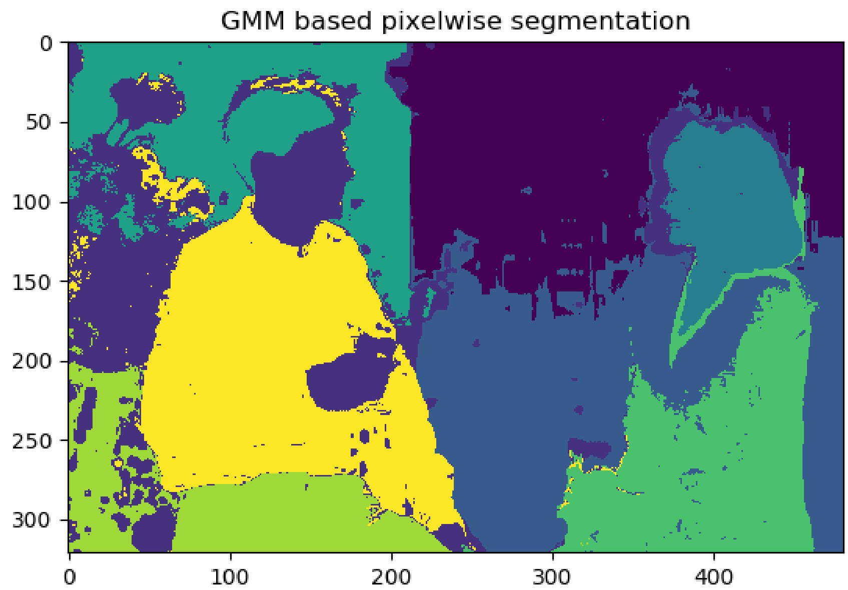
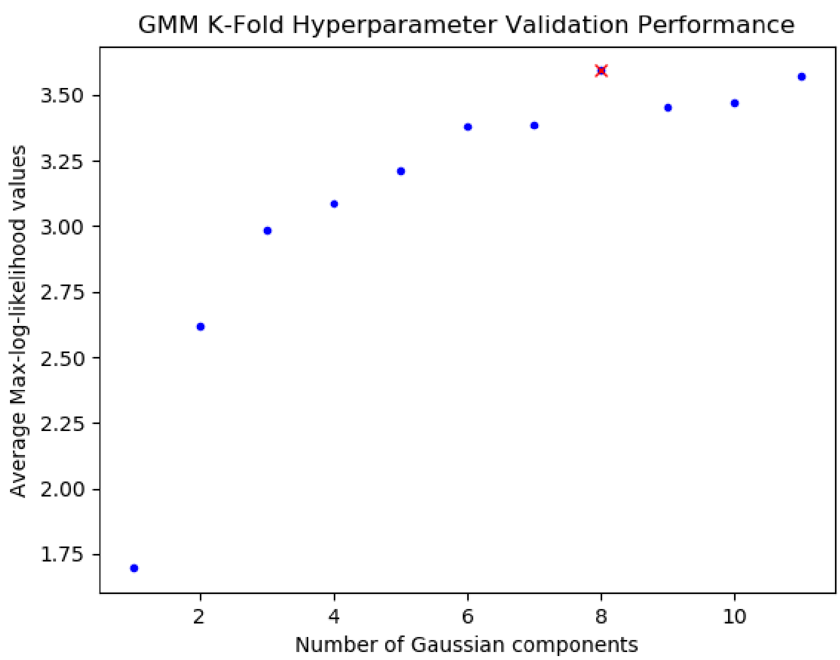


Image 3:



Below plot shows the average validation-log-likelihood for a varying number of gaussian components. For this image, the optimal number of gaussian components is **8.**(marked in red)



Appendix:

Question 1:

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from sklearn.model_selection import StratifiedKFold
from mpl_toolkits.axes_grid1 import make_axes_locatable
import keras
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
from keras.models import Sequential
from keras.layers import Dense
np.set_printoptions(suppress=True)
from tensorflow.keras.optimizers import SGD
from sklearn.svm import SVC

def gen_data(num_samples, priors):

    noise = np.random.multivariate_normal(noise_mu, noise_sigma, num_samples).T

    thetas = np.random.uniform(-np.pi, np.pi, num_samples)
    sample_data = np.random.uniform(0.0, 1.0, num_samples)
    num_cls1 = np.sum((sample_data <= priors[0]).astype('int'))
    num_cls2 = num_samples - num_cls1

    print('num_cls1: ',num_cls1)
    print('num_cls2: ',num_cls2)

    cos_thetas = np.cos(thetas).reshape((-1, 1))
    sine_thetas = np.sin(thetas).reshape((-1, 1))

    vecs = np.hstack((cos_thetas, sine_thetas)).T

    data_cls1 = r[0] * vecs[:, :num_cls1] + noise[:, :num_cls1]
    data_cls2 = r[1] * vecs[:, num_cls1:] + noise[:, num_cls1:]

    label_cls1 = np.zeros((1, num_cls1), dtype='int')
    label_cls2 = np.ones((1, num_cls2), dtype='int')

    data_wt_cls1 = np.vstack((data_cls1, label_cls1))
    data_wt_cls2 = np.vstack((data_cls2, label_cls2))

    return np.hstack((data_wt_cls1, data_wt_cls2))

def split_data(data_wt_labels, label_ids):
```

```
    samples = []

    for label_id in label_ids:
        class_ids = np.where(data_wt_labels[-1,:]==label_id)[0]
        cls_samples = data_wt_labels[:,class_ids]
        samples.append(cls_samples)

    return samples

def plot_data(data_wt_labels, label_ids, name):

    fig = plt.figure()
    ax = fig.add_subplot()

    samples = split_data(data_wt_labels, label_ids)

    colors = ['red', 'blue', 'green', 'brown']
    for label_id, sample in enumerate(samples):
        ax.scatter(sample[0, :], sample[1, :], s=5, color = colors[label_id], label = 'class ' +
str(label_id), marker='*')

    ax.set_title(name + ' data distribution')
    ax.set_xlabel('X')
    ax.set_ylabel('Y')

    plt.legend()
    plt.show()

def get_model(first_num_nodes, num_labels=1):

    sgd = SGD(lr=0.05, momentum=0.9)

    model = Sequential()

    # first layer
    fc1_act = Dense(units = first_num_nodes, kernel_initializer = 'random_uniform', activation =
'relu')
    model.add(fc1_act)

    # Second layer
    fc2_act = Dense(units = num_labels, kernel_initializer = 'random_uniform', activation =
'sigmoid')
    model.add(fc2_act)

    model.compile(optimizer=sgd, loss='binary_crossentropy', metrics = ['accuracy'])

    return model

def get_hp_values(num):
```

```
    hp_values = np.meshgrid(np.geomspace(0.05, 20, num), np.geomspace(0.05, 20, num))
    hp_values[0] = hp_values[0].reshape(num*num)
    hp_values[1] = hp_values[1].reshape(num*num)
    hp_values = np.vstack((hp_values[0], hp_values[1])).T

    return hp_values

def SVC_hyperparams(data_wt_labels, kfold):

    num_samples = data_wt_labels.shape[1] #(3, N)
    data_wt_labels = data_wt_labels[:, np.random.permutation(data_wt_labels.shape[1])]
#shuffle

    data = data_wt_labels[:2,:].T #(N, 2)
    labels = data_wt_labels[2,:].T

    print('data shape: ',data.shape)
    print('labels shape: ',labels.shape)

    num = 20
    hp_values = get_hp_values(num)
    print('hp_values: ',hp_values)

    hp_lst = []
    for C, kernel_width in hp_values:

        err_lst = []
        acc_lst = []
        skf = StratifiedKFold(n_splits=kfold, shuffle=False)
        print(C, kernel_width)

        for(val_idx, (train, val)) in enumerate(skf.split(data, labels)):

            # train
            train_data = data[train]
            train_labels = labels[train]

            # val
            val_data = data[val]
            val_labels = labels[val]

            # get model
            gamma = 1/(2*kernel_width**2)
            model = SVC(C=C, kernel='rbf', gamma=gamma)

            # train
            model.fit(train_data, train_labels)

            # validate
            predictions = model.predict(val_data)
```

```
            acc = np.sum(((predictions - val_labels) == 0).astype('int'))/val_data.shape[0]
            err = 1 - acc

            err_lst.append(err)
            acc_lst.append(acc)

        mean_err = np.mean(np.array(err_lst))
        std_err = np.std(np.array(err_lst))
        mean_acc = np.mean(np.array(acc_lst))

        print('num_samples:', num_samples, ' C: ',C, ' kernel_width: ',kernel_width,' mean error: ',
np.round(mean_err, 4), ' std error: ',np.round(std_err, 4), ' mean_acc: ', np.round(mean_acc,
4))
        hp_lst.append(mean_acc)

    hp_lst = np.array(hp_lst)
    desired_hp = hp_values[np.argmax(hp_lst)]
    print('desired_hp: ',desired_hp)
    max_acc = hp_lst[np.argmax(hp_lst)]
    min_err = 1 - max_acc
    print('max acc: ', max_acc)

    grid = np.meshgrid(np.geomspace(0.05, 20, num), np.geomspace(0.05, 20, num)
    )
    hp_lst_reshape = np.reshape(hp_lst, (num, num))

    fig, ax = plt.subplots()

    c = ax.pcolormesh(grid[0], grid[1], hp_lst_reshape, cmap='RdBu')
    ax.set_title("SVM K-Fold Hyperparameter Validation Performance")
    ax.set_xlabel("Overlap penalty weight")
    ax.set_ylabel("Gaussian kernel width")
    ax.axis([grid[0].min(), grid[0].max(), grid[1].min(), grid[1].max()])
    fig.colorbar(c, ax=ax, boundaries=np.linspace(0,1,10))
    ax.grid(True, color="crimson", lw=2)
    #plt.show()
    plt.savefig("SVM_hp.png")
    plt.close()

    return desired_hp

def MLP_hyperparams(data_wt_labels, kfold, num_perc_lst):

    num_samples = data_wt_labels.shape[1] #(3, N)
    data_wt_labels = data_wt_labels[:, np.random.permutation(data_wt_labels.shape[1])]
#shuffle

    data = data_wt_labels[:2,:].T #(N, 2)
    labels = data_wt_labels[2,:].T
```

```
    print('data shape: ',data.shape)
    print('labels shape: ',labels.shape)

    perc_lst = []
    perc_lst_acc = []
    for num_perc in num_perc_lst:

        err_lst = []
        acc_lst = []
        skf = StratifiedKFold(n_splits=kfold, shuffle=False)

        for(val_idx, (train, val)) in enumerate(skf.split(data, labels)):

            # train
            train_data = data[train]
            train_labels = labels[train]

            # val
            val_data = data[val]
            val_labels = labels[val]

            # get model
            model = get_model(num_perc)

            # train
            model.fit(train_data, train_labels, batch_size = 100, epochs = 300, verbose=0)

            # validate
            (err, accuracy) = model.evaluate(val_data, val_labels, verbose=0)

            print('num_samples:', num_samples,' num_perc: ',num_perc,' val idx: ', val_idx, ' error:
', np.round(err, 4), ' accuracy: ', np.round(accuracy, 4))
            err_lst.append(err)
            acc_lst.append(accuracy)

        mean_err = np.mean(np.array(err_lst))
        std_err = np.std(np.array(err_lst))
        mean_acc = np.mean(np.array(acc_lst))

        print('num_samples:', num_samples, ' num_perc: ',num_perc,' mean error: ',
np.round(mean_err, 4), ' std error: ',np.round(std_err, 4), ' mean_acc: ', mean_acc)
        perc_lst.append(mean_err)
        perc_lst_acc.append(mean_acc)

    perc_lst = np.array(perc_lst)
    print('pe for each perceptron: ', perc_lst)
    desired_num_perc = num_perc_lst[np.argmin(perc_lst)]

    plt.plot(num_perc_lst, perc_lst, '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(desired_num_perc, perc_lst[np.argmin(perc_lst)], 'rx')
    print("The best MLP accuracy was " + str(perc_lst_acc[np.argmin(perc_lst)]) + ".")
    #plt.show()
    plt.savefig("MLP_perc.png")

    return desired_num_perc

def train_kfoldMLP(train_wt_cls, test_wt_cls, kfold):

    num_train = train_wt_cls.shape[1]

    train_wt_cls = train_wt_cls[:, np.random.permutation(train_wt_cls.shape[1])] #shuffle
    train_data = train_wt_cls[:2,:].T #(N, 2)
    train_labels = train_wt_cls[2,:].T

    num_perc_lst = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11])*2

    print('train_wt_cls shape ',train_wt_cls.shape)
    # Model Order Selection
    desired_num_perc = MLP_hyperparams(train_wt_cls, kfold, num_perc_lst)

    # get model
    model = get_model(desired_num_perc)

    # train
    model.fit(train_data, train_labels, batch_size = 100, epochs = 300, verbose=0)

    print('model summary')
    print(model.summary())

    # validate
    test_data = test_wt_cls[:2,:].T #(N, 2)
    test_labels = test_wt_cls[2,:].T
    (val_err, val_acc) = model.evaluate(test_data, test_labels)

    print('num_samples: ',num_train,' desired_num_perc: ',desired_num_perc,' val_err: ',
val_err, ' val_acc: ', val_acc)

    plot_prediction(model, test_data, test_labels, 'MLP')

def train_kfoldSVC(train_wt_cls, test_wt_cls, kfold):

    num_train = train_wt_cls.shape[1]

    train_wt_cls = train_wt_cls[:, np.random.permutation(train_wt_cls.shape[1])] #shuffle
    train_data = train_wt_cls[:2,:].T #(N, 2)
    train_labels = train_wt_cls[2,:].T
```

```
    # Model Order Selection
    desired_hp = SVC_hyperparams(train_wt_cls, kfold)
    print('desired params: ',desired_hp)

    desired_C, desired_kernel_width = desired_hp[0], desired_hp[1]

    # get model
    model = SVC(C=desired_C, kernel='rbf', gamma=1/(2*desired_kernel_width**2))

    # train
    model.fit(train_data, train_labels)

    # validate
    test_data = test_wt_cls[:2,:].T #(N, 2)
    test_labels = test_wt_cls[2,:].T
    predictions = model.predict(test_data)
    val_acc = np.sum(((predictions - test_labels) == 0).astype('int'))/test_data.shape[0]
    val_err = 1 - val_acc

    print('num_samples: ',num_train,' C: ',desired_C,' kernel_width: ',desired_kernel_width,'
val_err: ', val_err, ' val_acc: ', val_acc)

    plot_prediction(model, test_data, test_labels, 'SVC')

def plot_prediction(model, test_data, test_labels, method):

    predictions = np.squeeze(model.predict(test_data))
    correct = np.array(np.squeeze((np.round(predictions) == test_labels).nonzero()))
    incorrect = np.array(np.squeeze((np.round(predictions) != test_labels).nonzero()))

    plt.plot(test_data[correct][:,0],
            test_data[correct][:,1],
            'k.', alpha=0.25)
    plt.plot(test_data[incorrect][:,0],
            test_data[incorrect][:,1],
            'r.', alpha=0.25)

    plt.title(method + ' Classification Performance')
    #plt.title(method + ' Confidence Contour')
    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)))
    # CS = plt.contour(gridpoints[0], gridpoints[1], contour_values)
    # CB = plt.colorbar(CS, shrink=0.8, extend='both', cmap='magma')
```

```
    plt.contourf(gridpoints[0], gridpoints[1], contour_values, levels=1)
    plt.colorbar(cmap='magma')

    plt.show()
    #plt.savefig(method + "_plt_pred.png")

if __name__ == "__main__":

    priors = [0.4, 0.6]
    label_ids = [0, 1]
    num_train_samples = 1000
    num_test_samples = 10000
    noise_mu = [0, 0]
    noise_sigma = np.eye(2, dtype=float)
    r = [2, 4]
    kfold = 10

    ## train
    train_wt_cls = gen_data(num_train_samples, priors)
    plot_data(train_wt_cls, label_ids, 'Training')

    ## test
    test_wt_cls = gen_data(num_test_samples, priors)
    plot_data(test_wt_cls, label_ids, 'Testing')

    ## Train MLP
    train_kfoldMLP(train_wt_cls, test_wt_cls, kfold)

    ## Train SVC
    train_kfoldSVC(train_wt_cls, test_wt_cls, kfold)
```

Question 2

```
import numpy as np
import cv2
from sklearn.mixture import GaussianMixture
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import normalize
from scipy.stats import multivariate_normal
import matplotlib.pyplot as plt

def train_val_split(data, idx, kfold):

    num_samples = data.shape[0]
    num_samples_batch = num_samples//kfold

    train_data = np.concatenate((data[0:idx*num_samples_batch],
```

```
data[(idx+1)*num_samples_batch:]), axis=0)
    val_data = data[idx*num_samples_batch : (idx+1)*num_samples_batch]

    return train_data, val_data

def MOS(data, num_gmm_lst, kfold):

    num_samples = data.shape[0]
    gmm_mls = []

    for num_gmm in num_gmm_lst:

        max_likelihoods = []
        for val_idx in range(kfold):

            # train test split
            train_data, val_data = train_val_split(data, val_idx, kfold)

            GMM = GaussianMixture(num_gmm, covariance_type='full',
                    random_state=0)

            GMM.fit(train_data)

            max_likelihood = GMM.score(val_data)
            num_val_samples = val_data.shape[0]
            max_likelihoods.append(num_val_samples * max_likelihood)

            print('val idx: ', val_idx, ' num_val_samples: ', num_val_samples, ' num_gmm: ',
num_gmm, ' mls: ', np.round(max_likelihood, 3))

        mean_max_likelihoods = np.sum(max_likelihoods)/num_samples
        gmm_mls.append(mean_max_likelihoods)
        print('num_gmm: ', num_gmm, ' mean_mle: ',np.round(mean_max_likelihoods, 3))

    desired_num_gmm = num_gmm_lst[np.argmax(gmm_mls)]

    return desired_num_gmm

def get_feature_vector(img):

    h, w = img.shape[:2]
    num_pixels = h * w

    feat_vec = np.zeros((num_pixels, 5), dtype='float')

    for row in range(h):
        for col in range(w):
            feat_vec[row*w + col, 0] = row
            feat_vec[row*w + col, 1] = col
            feat_vec[row*w + col, 2] = img[row, col, 2] #red
```

```
            feat_vec[row*w + col, 3] = img[row, col, 1] #green
            feat_vec[row*w + col, 4] = img[row, col, 0] #blue

    norm_feat_vec = normalize(feat_vec, axis=0, norm='max')

    return norm_feat_vec

if __name__ == "__main__":

    img_path = '157055.jpg'

    num_gmm_lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
    kfold = 10

    img = cv2.imread(img_path)
    print('img shape ',img.shape)
    norm_feat_vec = get_feature_vector(img)
    print('norm_feat_vec: ',norm_feat_vec)

    desired_num_gmm = MOS(norm_feat_vec, num_gmm_lst, kfold)

    print('desired_num_gmm: ',desired_num_gmm)

    model = GaussianMixture(desired_num_gmm, covariance_type='full',
                  random_state=0, init_params='kmeans', max_iter=100)
    model.fit(norm_feat_vec)

    prediction = np.zeros((norm_feat_vec.shape[0], desired_num_gmm))
    for i in range(desired_num_gmm):
        pdf = multivariate_normal.pdf(norm_feat_vec,
mean=model.means_[i,:],cov=model.covariances_[i,:,:])
        prediction[:, i] = model.weights_[i] * pdf

    prediction = np.argmax(prediction, axis=1)
    print('prediction ', prediction,prediction.shape)
    prediction = prediction.reshape((img.shape[0], img.shape[1]))
    plt.imshow(prediction)
    plt.show()
```

References:

1. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
2. https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html
3. https://medium.com/all-things-ai/in-depth-parameter-tuning-for-svc-758215394769