EECE5644 - Assignment 4

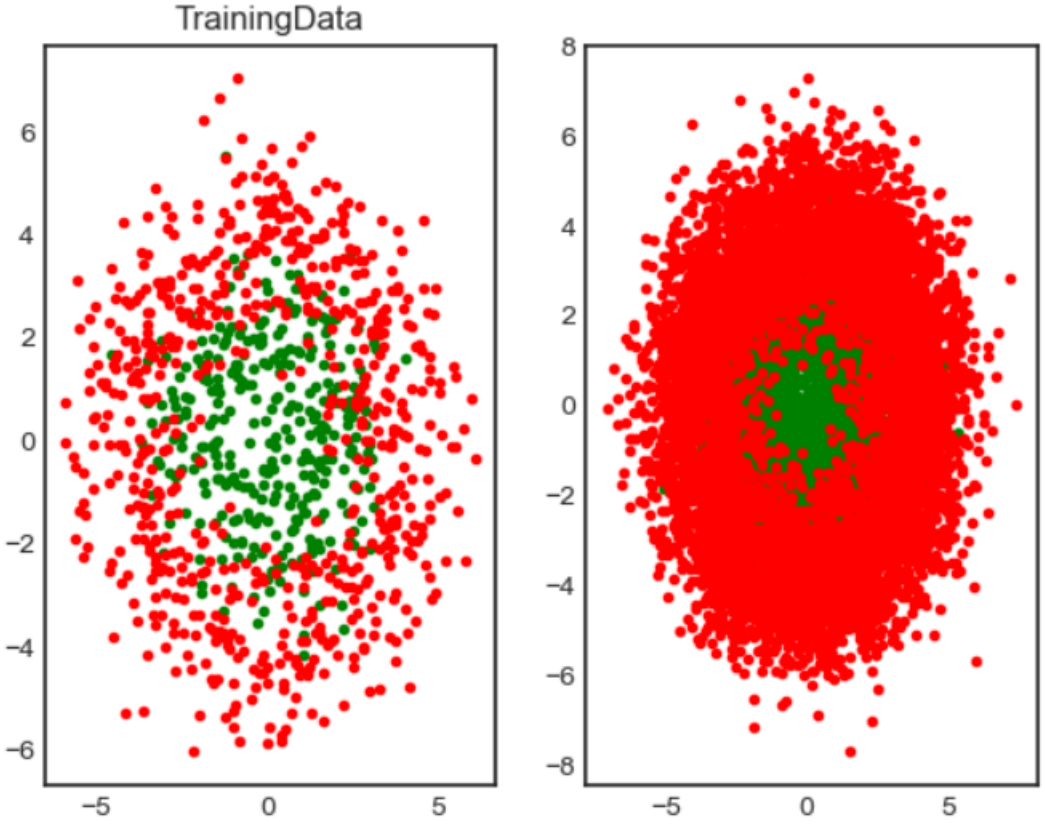
Name: Bhuvan Karthik Channagiri

NU ID: 002825947

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.

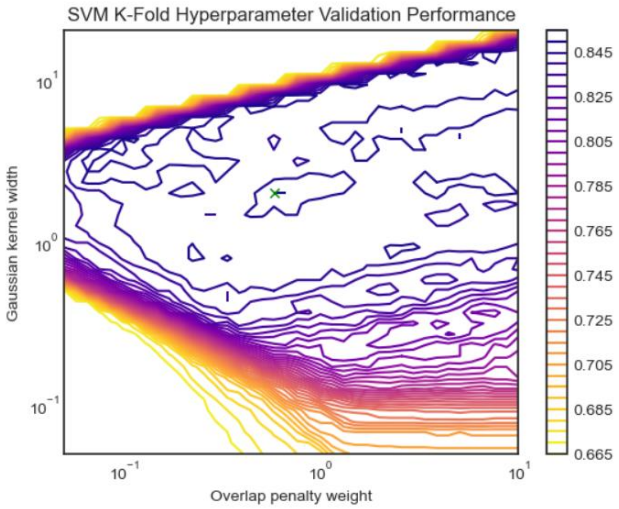
All data for class l ∈ {−1, +1} where class C is given as $\mathbf{x} = r_l \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} + \mathbf{n}$ where $\theta \sim$Uniform[$-\pi, \pi$] and n ∼ N(0, σ 2I). Use r − 1 = 2, r+1 = 4, σ = 1. Class priors are 0.4 for class 0 and 0.6 for class 1.
Mean class 0 = 2 , class 1= 4 , Sigma=1 .



During K-Fold validation for SVM, a range of 40 values for the overlap penalty weight and Gaussian kernel width were explored within the intervals [0.5, 10] and [0.5, 20], respectively. The selection process involved ten-fold cross-validation to identify the optimal hyperparameters: the box constraint hyperparameter \(C\) and the Gaussian kernel width hyperparameter \(\sigma\).

In the 10-fold cross validation we would perform a fitting procedure a total of 10 times, in which the dataset is divided into 10 folds or subsets. Each iteration selects 90% of the dataset for training and the remaining 10% for the validation. With each iteration we select a different 90% of the dataset at random and take the remaining 10% for the validation and we calculate the minimum probability of error across the cross validation runs. Once we conclude on the number to perceptron to use then we can run it again on the full dataset and calculate the probability of error.

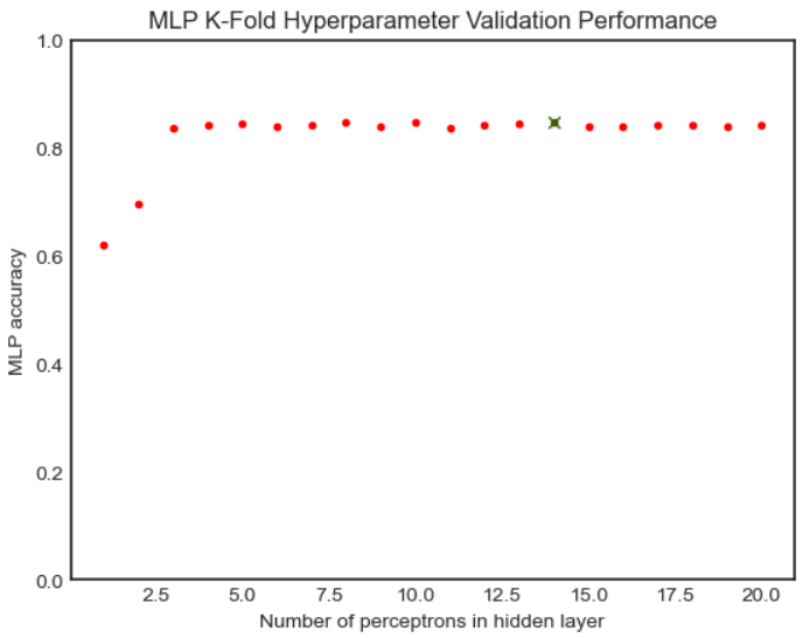
The criteria for selecting the best hyperparameter values were based on minimizing the average cross-validation probability of error. Once the hyperparameters were determined, the entire training set was utilized to train the model using the chosen hyperparameters. Subsequently, the performance of this trained model was assessed on the test dataset. The contour plot below illustrates the accuracy achieved for each tested combination of hyperparameters.



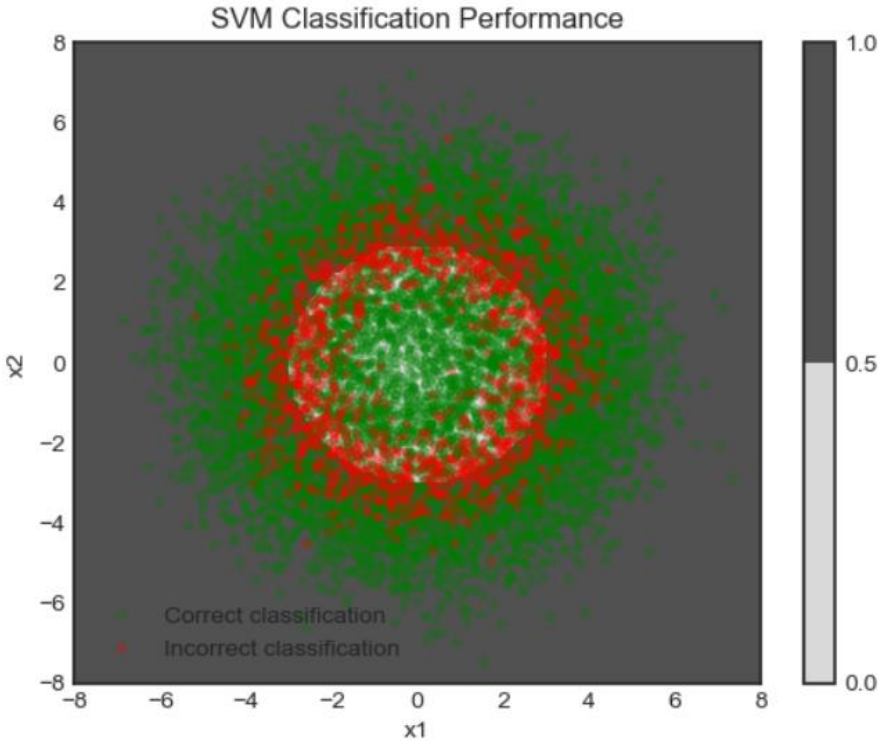
The maximum achieved accuracy occurred using an overlap penalty weight of 0.3836909370840584 and a gaussian kernel width of 6.8232861842018995, as marked with a green ‘x’ on the plot above. The best SVM Accuracy while training is 0.845. This combination produced an average accuracy of 0.845 on the 10 K- fold validation partitions. The depicted plot exhibits ascending flat plateaus, both at the lower and upper accuracy bounds. When the overlap penalty weight and kernel width are sufficiently low, samples are effectively positioned "too far" from each other, hindering the formation of meaningful clustered groups. Conversely, with a low overlap penalty weight and a large kernel width, samples become "too close" to each other, making meaningful separation unattainable. Striking the right balance between these two parameters allows for the derivation of an appropriate decision boundary. However, this boundary proves ineffective in accurately classifying samples within the overlapping Gaussian distributions.
Here, the classification boundary seems circular as it is expected to be for this mode of data generation.

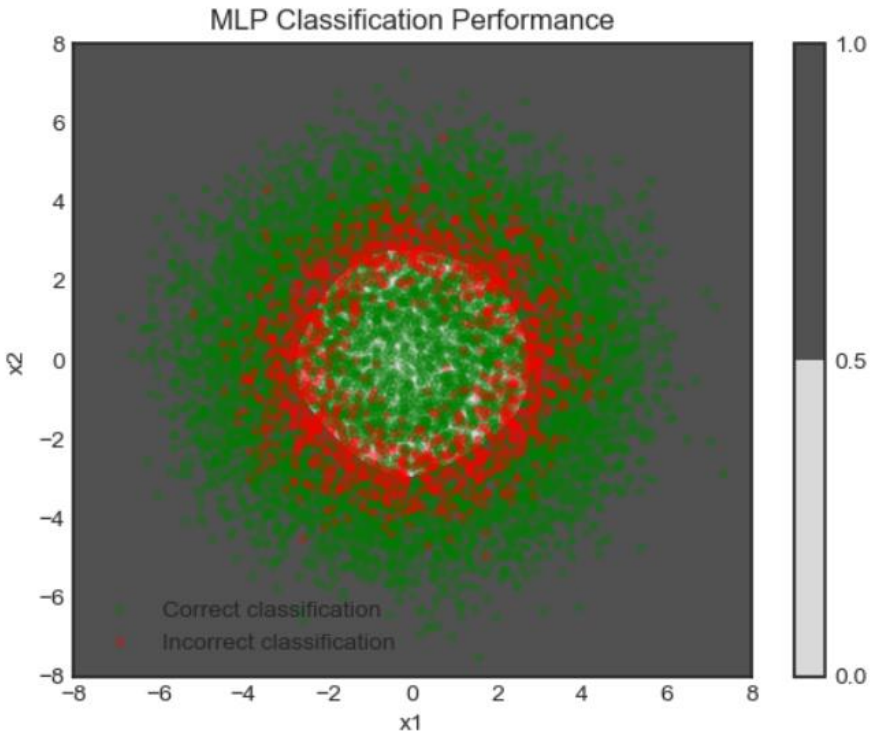
During K-fold validation for the MLP(Multi layer Percepton) model, up to 14 perceptons were tested in the hidden layer. The accuracy achieved for each model is displayed in the plot below.

From the figure given below, we can see that the best MLP accuracy (K fold hyperparameter) was achieved with 14 perceptons.



The maximum achieved accuracy occurred using 14 perceptons in the hidden layer, as marked with a green 'x' on the plot. **The best MLP accuracy while training is 0.8338000178337097.** This configuration produced an average accuracy of 0.845 on the 10 K-fold validation partitions. After the 3 parameters, the percepton tends to remain constant generally hence, we consider the accuracy is achieved with the help of 14 perceptons. Similar to the SVM results, there is a smooth plateau that occurs at the maximum accuracy for the data in question. The optimal model selected by K-fold validation has 14 perceptons but other quantities greater than 2 perform about the same as well. The given below figure has been generated by training the optimal SVM model selected by K-fold validation on the entire test dataset. The red color denotes the incorrect classification while the green color denotes the correct classification.



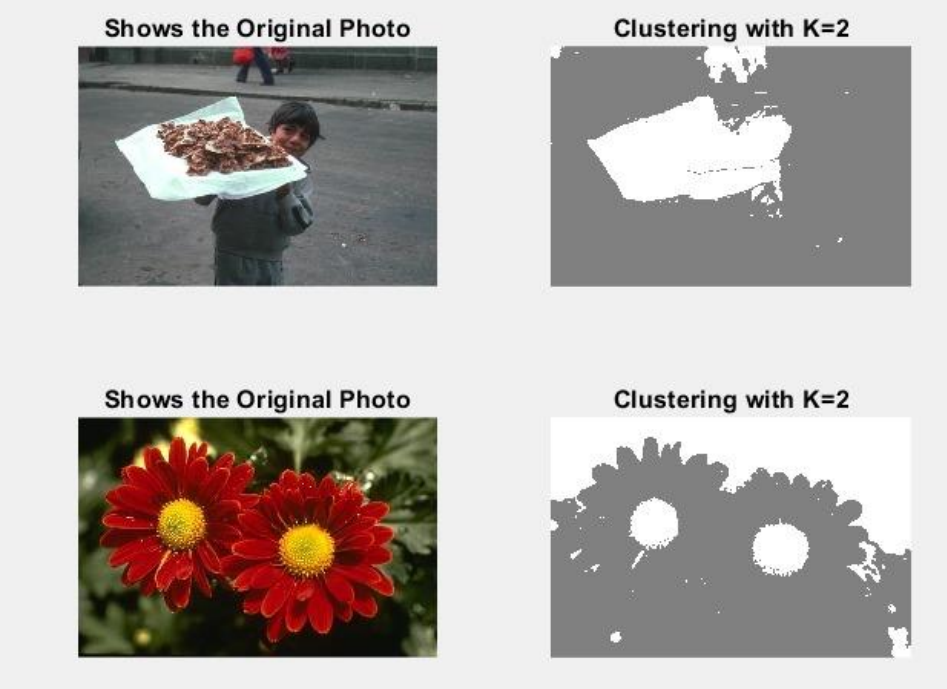


The model above was fit with an accuracy of **0.833800**. Here we can see that the classification boundary is roughly close to circular.
The test dataset was fit by the SVM model with an accuracy of **0.8368.**
The test dataset was fit by the MLP model with an accuracy of **0.8338000178337097**.

The MLP classifier was able to train the data with slightly better accuracy than the SVM classifier even though it takes a longer time as compared to the SVM classifier. The boundary created by the MLP is more smoother unlike the jagged boundary of the SVM model. But both models are close to maximum accuracy of up to 85% (or 15% probability of error)

Question – 2:

In this section, we use GMM-based clustering to segment colour image. The image below shows the RGB colour image and its corresponding grayscale image after clustering with K = 2 components.



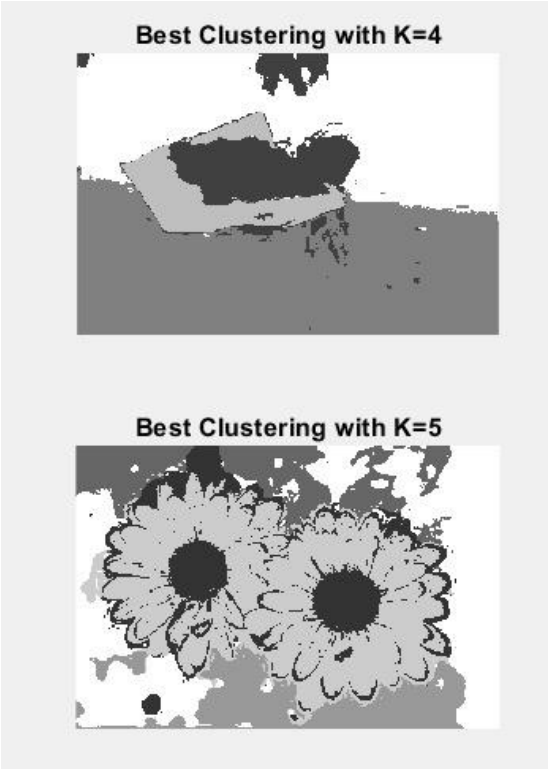
The following process was performed on the colour images:

1. For preprocessing, generate a 5-dimensional feature array (comprising of row index, column index, r, G, B values)
2. For each pixel:
    - Append row index, column index, red, green and blue values to a feature array
    - Normalize each feature to [0, 1], i.e., the feature vectors fit in the 5-dimensional hypercube
    - Run segmentation algorithm on normalized feature vectors

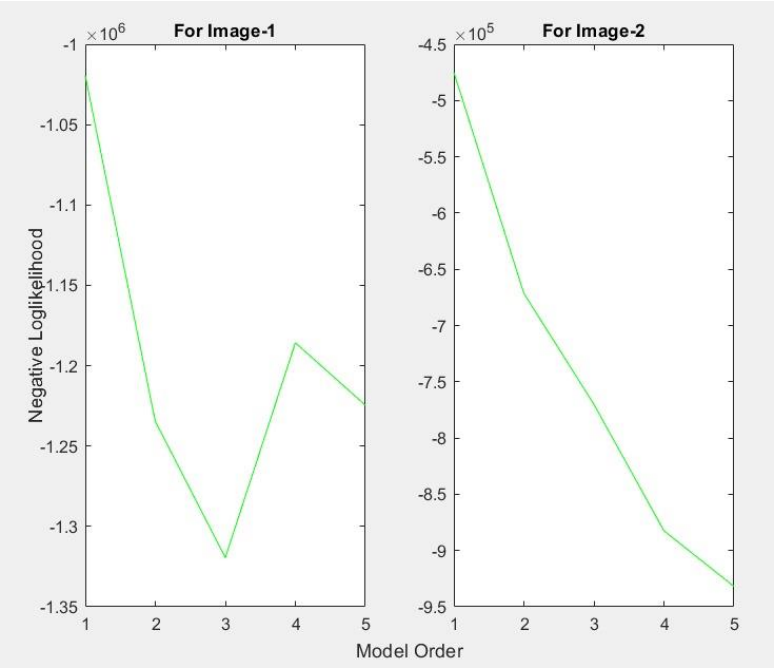
We then fit a GMM to our data with 2 components.

To find the optimal or best GMM order for clustering we perform MLP eatimation using 10-fold cross validation. Once the optimal number of components is selected, the resulting GMM is fit to the data. The GMM components are used as class conditional PDFs so we can use MAP classification to cluster different parts of the image together.

The below image shows the optimal number of GMM components selected was 4 for image-1 and 5 for image-2.



In the below figure we see the results of the cross validation for the images. The average log likelihood for each model order can be seen, and we observe that for **Image-1** it has the greatest log likelihood at model order 4 of the component values tested, similarly it is 5 for **Image-2**.



Appendix:

Question 1:

```
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
import os

os.environ["CUDA_VISIBLE_DEVICES"] = "0"

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).non
zero())][0,:,0],
             TrainingData_features[np.array((TrainingData_labels==0).nonz
ero())][0,:,1],
             'g.')
    plt.plot(TrainingData_features[np.array((TrainingData_labels==1).non
zero())][0,:,0],
             TrainingData_features[np.array((TrainingData_labels==1).nonz
ero())][0,:,1],
             'r.')
    plt.title('TrainingData')

    plt.subplot(1,2,2)

    plt.plot(TestingData_features[np.array((TestingData_labels==0).nonze
ro())][0,:,0],
             TestingData_features[np.array((TestingData_labels==0).nonzer
o())][0,:,1],
             'g.')
    plt.plot(TestingData_features[np.array((TestingData_labels==1).nonze
ro())][0,:,0],
             TestingData_features[np.array((TestingData_labels==1).nonzer
o())][0,:,1],
             'r.')
    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, 'gx')
    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, 'r.')
    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], 'gx')
    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],
             'g.', 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:

```
%% Initialize

clear all; close all;

f{1,1} =
"https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/BSDS300/html/images/plain/normal/color/90076.jpg";

f{1,2} =
"https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/BSDS300/html/images/plain/normal/color/124084.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(:)'];

   for d = 1:D

      imdatad = imdata(:,:,d);

      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,'-g');
    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');
%% 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
```

Credits:

- Professor Deniz's Notes
- GitHub