EECE5644 Assignment4

Name: Liangshe Li
NUID: 001586357

Question 1 (60%)
Train and test Support Vector Machine (SVM) and Multi-layer Perceptron (MLP) classifiers that aim for minimum probability of classification error (i.e. we are using 0-1 loss; all error instances are equally bad). You may use a trusted implementation of training, validation, and testing in your choice of programming language. The SVM should use a Gaussian (sometimes called radial-basis) kernel. The MLP should be a single-hidden layer model with your choice of activation functions for all perceptrons.
Generate 1000 independent and identically distributed (iid) samples for training and 10000 iid samples for testing. All data for class l ∈ {−1,+1} should be generated as follows:

$$\mathbf{x} = r_l \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} + \mathbf{n} \tag{1}$$

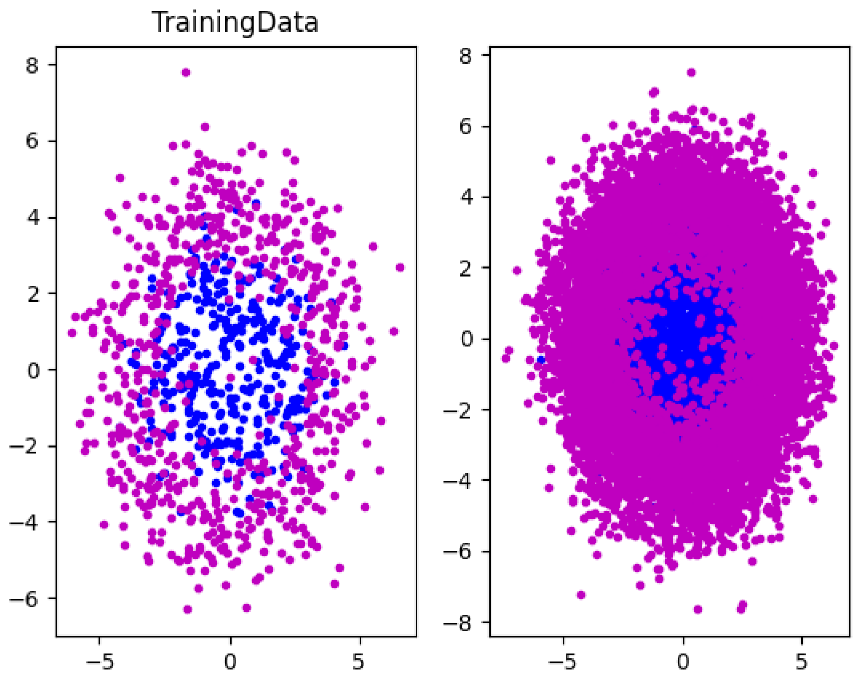
where θ ~Uni f orm[−π,π] and n ~ N(0,σ2I). Use r−1 = 2, r+1 = 4,σ = 1.

Note: The two class sample sets will be highly overlapping two concentric disks, and due to angular symmetry, we anticipate the best classification boundary to be a circle between the two disks. Your SVM and MLP models will try to approximate it. Since the optimal boundary is expected to be a quadratic curve, quadratic polynomial activation functions in the hidden layer of the MLP may be considered as to be an appropriate modeling choice. If you have time (optional, not needed for assignment), experiment with different activation function selections to see the effect of this choice.
Use the training data with 10-fold cross-validation to determine the best hyperparameters (box constraints parameter and Gaussian kernel width for the SVM, number of perceptrons in the hidden layer for the MLP). Once these hyperparameters are set, train your final SVM and MLP classifier using the entire training data set. Apply your trained SVM and MLP classifiers to the test data set and estimate the probability of error from this data set.
Report the following: (1) visual and numerical demonstrations of the K-fold cross-validation process indicating how the hyperparameters for SVM and MLP classifiers are set; (2) visual and numerical demonstrations of the performance of your SVM and MLP classifiers on the test data set. It is your responsibility to figure out how to present your results in a convincing fashion to indicate the quality of training procedure execution, and the test performance estimate.
Hint: For hyperparameter selection, you may show the performance estimates for various choices and indicate where the best result is achieved. For test performance, you may show the data and classification boundary superimposed, along with an estimated probability of error from the samples. Modify and supplement these ideas as you see appropriate.

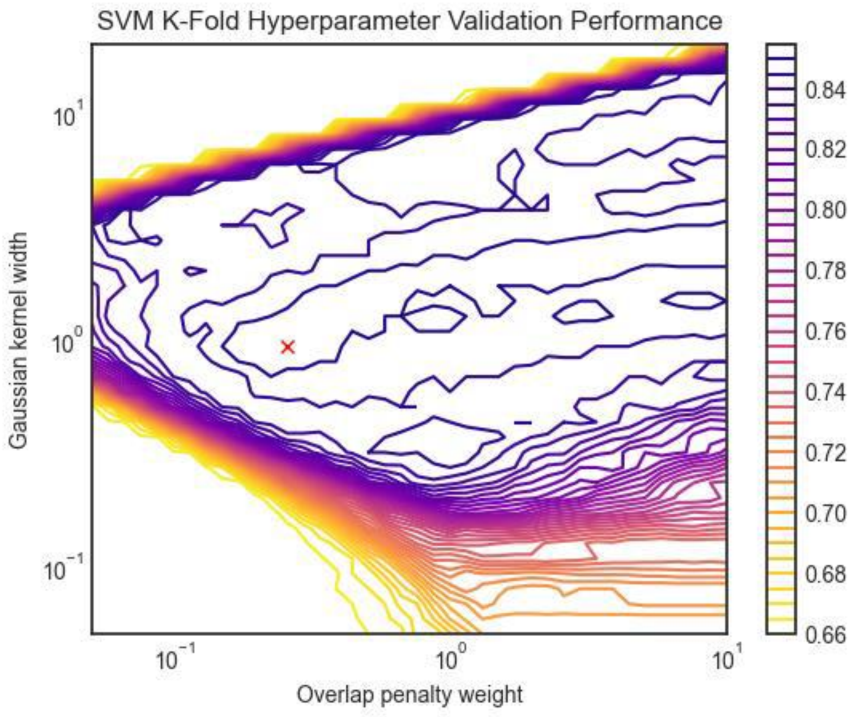
Answer:

The data I use was generated with $r_{-1}=2, r_1=4, \sigma = 1$. And the class prior is [0.65,0.35]

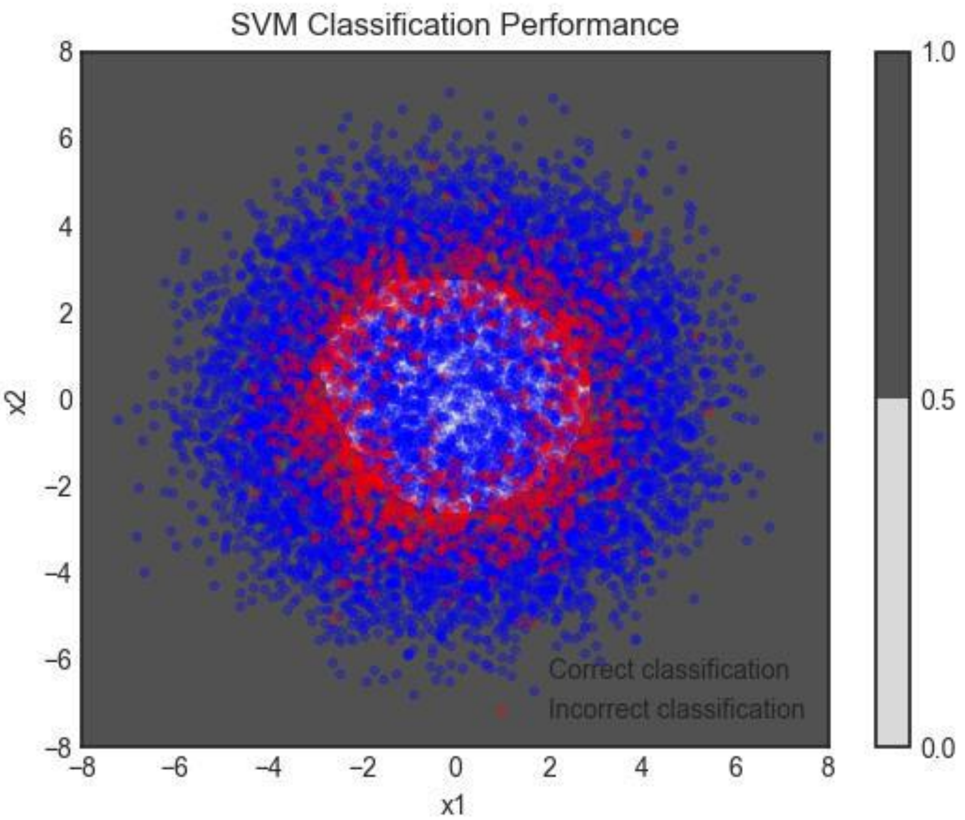
The data generated is as follows:



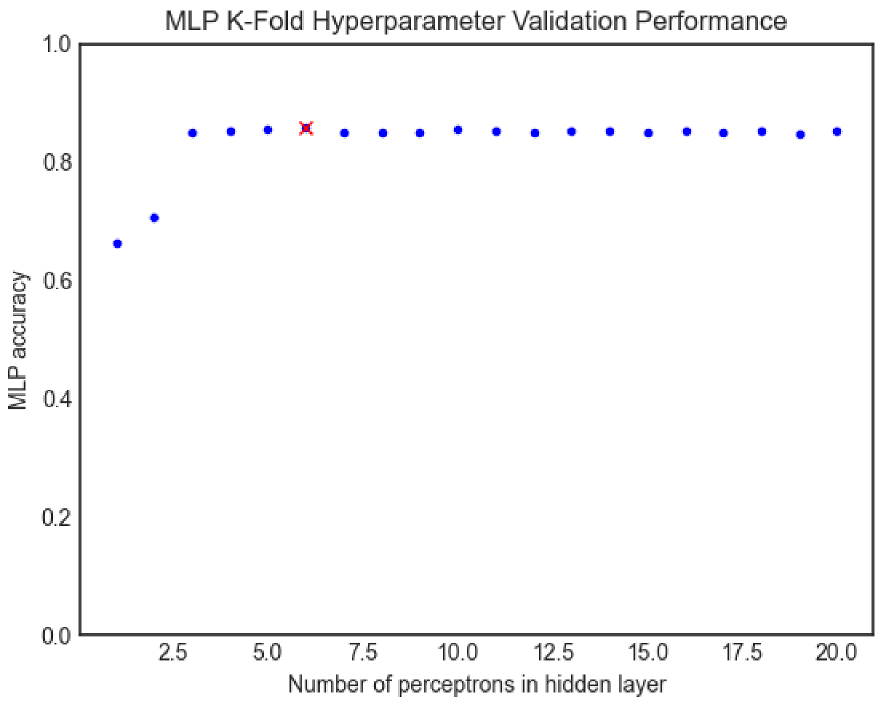
During K-Fold validation for the SVM, 40 values of the overlap penalty weight and Gaussian kernel width were tested, respectively. The accuracy achieved for a model with each tested combination is shown in the contour plot below:



The maximum achieved accuracy occurred using an overlap penalty weight of 0.2552 and a Gaussian kernel width of 0.9260, as marked with a red 'x' on the plot above. This combination produced an average accuracy of 0.8469 on the 10 K-Fold validation partitions.
The plot above has 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 below figure was generated by training the optimal SVM model selected by K-Fold validation on the entire test dataset.



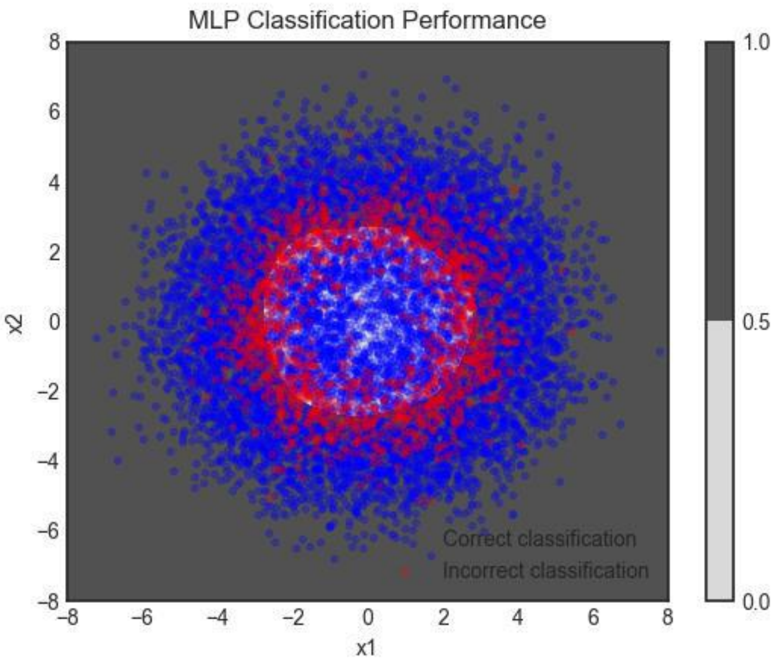
The model above was fit with an accuracy of 0.8427. The classification boundary appears roughly circular, as is to be expected according to the method of data generation.
During K-Fold validation for the MLP model, up to 20 perceptrons were tested in the hidden layer. The accuracy achieved for each model is shown in the plot below.



The maximum achieved accuracy occurred using 6 perceptrons in the hidden layer, as marked with a red 'x' on the plot above. This configuration produced an average accuracy of 0.8570 on the 10 K-Fold validation partitions.

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 6 perceptrons, but other quantities greater than 2 perform about the same as well.

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

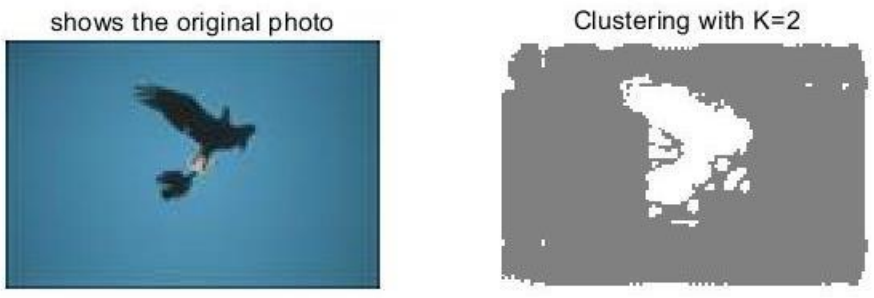


The model above was fit with an accuracy of 0.8482. Once again, the classification boundary appears roughly circular.
The MLP classifier was able to achieve slightly better accuracy than the SVM classifier, although it takes much longer to train. Visually, the smoother boundary generated by the MLP model 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).

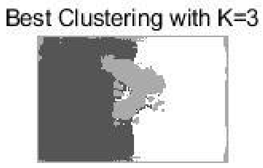
Question 2 (40%)
In this question, you will use GMM-based clustering to segment a color image. Pick your color image from this dataset:
https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/BSDS300/html/dataset/images.html
As preprocessing, for each pixel, generate a 5-dimensional feature vector as follows: (1) append row index, column index, red value, green value, blue value for each pixel into a raw feature vector; (2) normalize each feature entry individually to the interval [0,1], so that all of the feature vectors representing every pixel in an image fit into the 5-dimensional unit-hypercube.
Fit a Gaussian Mixture Model to these normalized feature vectors representing the pixels of the image. To fit the GMM, use maximum likelihood parameter estimation and 10-fold cross validation (with maximum average validation-log-likelihood as the objective) for model order selection.
Once you have identified the best GMM for your feature ectors, assign the most likely component label to each pixel by evaluating component label posterior probabilities for each feature vector according to your GMM. Present the original image and your GMM-based segmentation labels assigned to each pixel side by side for easy visual assessment of your segmentation outcome. If using grayscale values as segment/component labels, please uniformly distribute them between min/max grayscale values to have good contrast in the label image.
Hint: If the image has too many pixels for your available computational power, you may down-sample the image to reduce overall computational needs).

Answer:
Image I pick is: “135069.jpg”.
For this image, the following steps were performed. As preprocessing to generate a 5-dimensional feature array the following was performed for each pixel: (1) appended row index, column index, red value, green value, and blue value to a raw feature vector, (2) normalize each feature entry individually to the interval [0,1] so that all of the feature vectors representing every pixel in an image fit in the 5-dimensional hypercube. All segmentation algorithms operated on these normalized feature vectors.
Following preprocessing maximum likelihood parameter estimation was used to fit a GMM with 2-components to each image. This GMM was then used to segment the image into two parts to separate the foreground and the background. Then 10-fold cross-validation was used

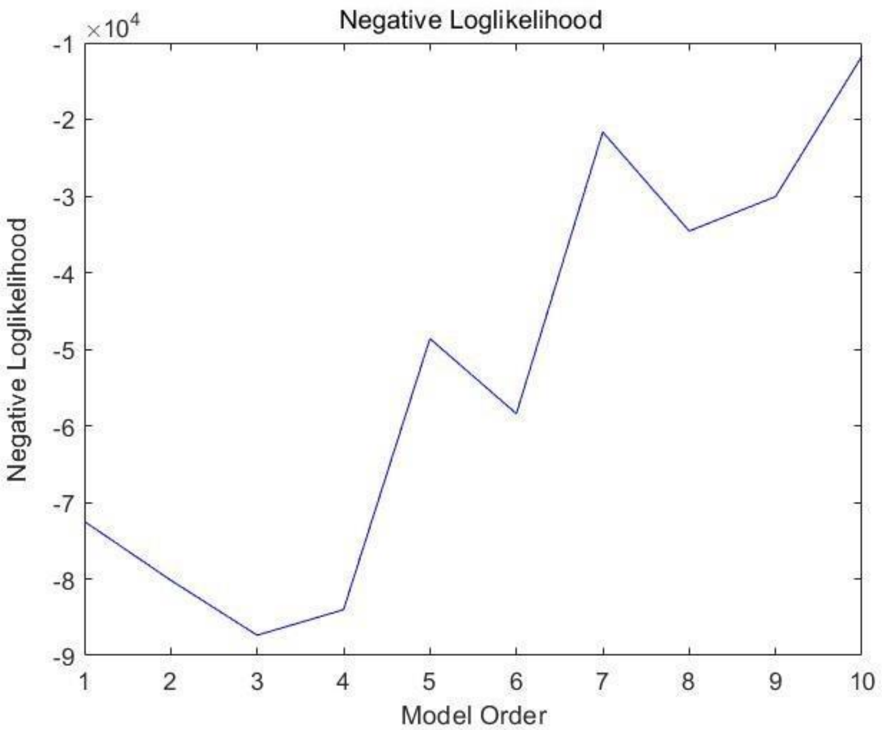
to identify the number of Gaussian components that produced the maximum average validation log-likelihood. After the estimated optimal number of Gaussian components was selected, a GMM with that number of components was fit to the image data. This new GMM model was then used to segment the image with the number of segments equal to the number of components in the GMM. For clustering the GMM components were used as class/cluster conditional pdfs and used to assign cluster labels using the MAP classification rule.
The original image along with the segmented versions are shown in the following graph:



For the image the segmented images include the segmented image for 2 GMMs which was specified in the problem and the number of GMMs that maximized the log likelihood as determined by cross-validation which is 3.



The next figure shows the negative loglikelihood results for the image. Since it was negative data, the actual average likelihood for this image was largest in the range from 2 to 4 GMMs and then decreased as additional GMMs were added.



Codes

Question1(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
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).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)

Question2(Matlab)

```
    imdata  = imread(file(i));
    figure(1), subplot(1, 2, 1*2-1),
    imshow(imdata);
    title("shows the original photo"); hold on;
    [R,C,D] = size(imdata); N = R*C; imdata = douclear all; close all;
%%========================Setup========================%%
file = ["135069.jpg"];
K = 10;
M = 10;
n = size(file, 1);

%%========================Segmentation========================%%
for i=1:length(file)
ble(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, 1*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,1,1),
    imshow(uint8(li*255/mini));
    title(strcat("Best Clustering with K=", num2str(mini)));
    fig=figure(3);
    subplot(1,n,1), plot(ab,'-b');
end

rst = axes(fig, 'visible', 'off');
rst.Title.Visible='on';
rst.XLabel.Visible='on';
rst.YLabel.Visible='on';
ylabel(rst,'Negative Loglikelihood');
xlabel(rst,'Model Order');
title(rst,['Negative Loglikelihood']);

%%========================Question2 functions========================%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
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
```

```matlab
            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
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```