EECE5644 HW4

CONTENT

[Question 1 1](#_Toc88070677)

[Question 2 6](#_Toc88070678)

[Appendix code for Question 1 8](#_Toc88070679)

[Appendix code for Question 2 17](#_Toc88070680)

# Question 1

The data was generated with And the class prior is [0.65,0.35]

图表, 散点图

描述已自动生成

*Figure 1: Data distribution*

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

图表

描述已自动生成

*Figure 2: SVM validation performance*

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.

图片包含 图形用户界面

描述已自动生成

*Figure 3: SVM classification performance*

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.

图表, 散点图

描述已自动生成

*Figure 4: MLP validation performance*

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.

图片包含 图表

描述已自动生成

*Figure 5: MLP classification performance*

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

Image was picked: “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 Figure 6. 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.

图形用户界面

中度可信度描述已自动生成

*Figure 6: Image and Segmentation Result with K=2*

图示

低可信度描述已自动生成

*Figure 7: Best Segmentation Result*

Figure 8 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.

图表, 折线图

描述已自动生成

*Figure 8: Negative LogLikelihood vs. Number of GMMs for the Image*

Appendix code for Question 1

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%EECE5644 Fall 2021  
% Wang Yinan 001530926 | HW4  
%%=========================Question 1 py=========================%%  
% Code help and example from Prof.Deniz  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
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)  
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)

Appendix code for Question 2

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%EECE5644 Fall 2021  
% Wang Yinan 001530926 | HW4  
%%=========================Question 2=========================%%  
% Code help and example from Prof.Deniz  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
clear all; close all;  
%%=========================Setup=========================%%  
file = ["135069.jpg"];  
K = 10;  
M = 10;  
n = size(file, 1);

%%=========================Segmentation=========================%%

for i=1:length(file)  
    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 = 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, 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  
            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  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%