EECE 5644: Machine Learning / Pattern Recognition

Assignment 4

Jiayun Xin

NUID: 001563582

College of Engineering

Northeastern University Boston, Massachusetts

Spring, 2022

Q1

Code for this question can be found in Appendix A

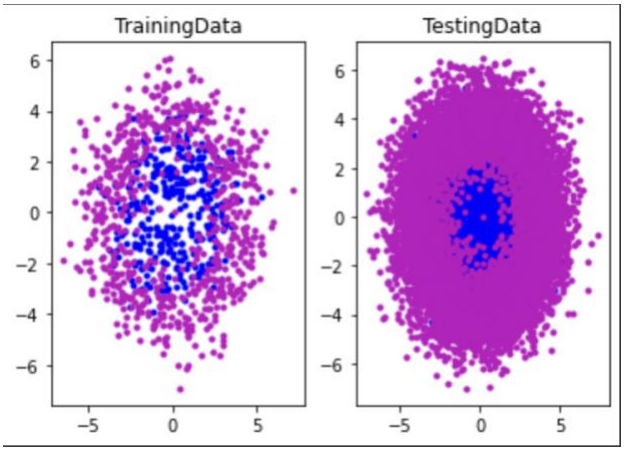

Figure 1

The figure 1 shows the basic dataset.

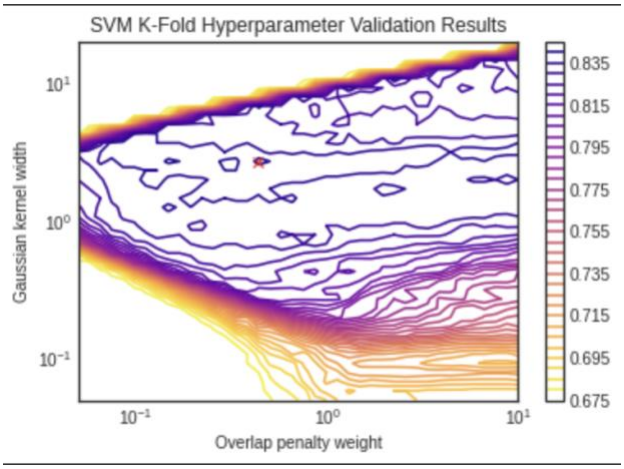

Figure 2

During K-Fold validation of the SVM, there are 40 values of the overlap penalty weight tested in the ranges [0.5, 10] and the Gaussian kernel width was tested in the ranges [0.5, 20]. The accuracy values relevant to overlap penalty weight and Gaussian kernel width are shown in figure 2 with a flat plateau. As we can see, the red ‘x’ marked in the plot is the point with highest accuracy of 0.842. The best SVM accuracy was achieved with an overlap penalty weight of 0.43952371232482884 and a Gaussian kernel width of 2.7144176165949068.
The combination connecting low overlap penalty weight and low kernel width causes data far away from each other. The combination connection low overlap penalty weight and high kernel width results in data too close from each other. A good combination between overlap penalty weight and kernel width decides the quality of decision boundary.

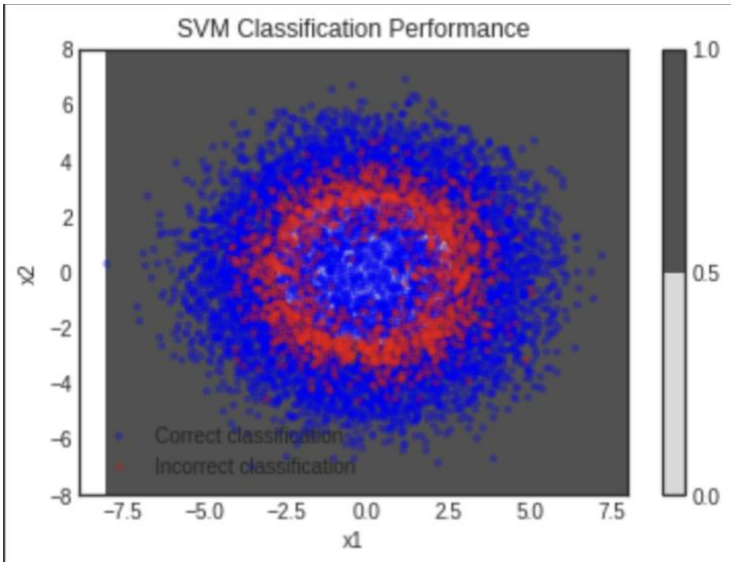

Figure 3

The figure 3 shows the SVM classification results with an accuracy of 0.8343. The boundary is a rough circle.

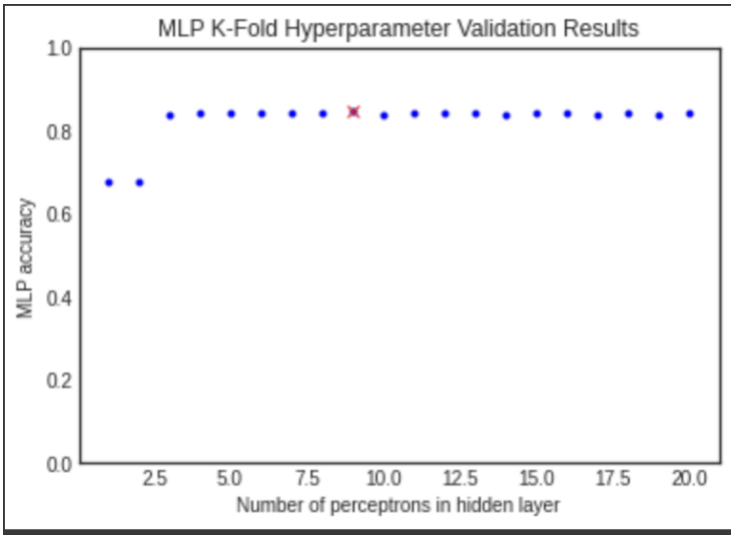

Figure 4

During K-Fold validation of the SVM, there are 20 perceptrons tested in the hidden layer. The MLP classification accuracy as numbers of perceptrons increase is shown in figure 4. The red ‘x’ marked in the plot is the point with highest accuracy of 0.846. The best MLP accuracy was achieved with 9 perceptrons. When the number of perceptrons is larger than 2, the MLP accuracy almost keeps stable.

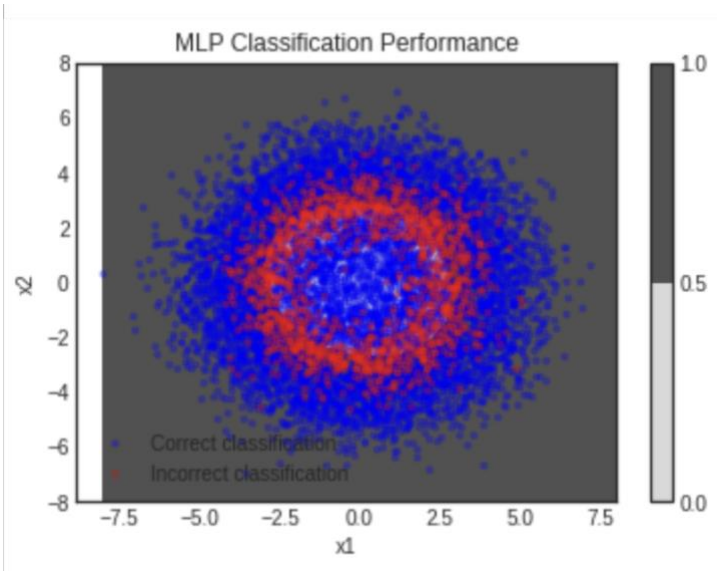

Figure 5

The figure 5 shows the SVM classification results with an accuracy of 0.8365. The boundary is also a rough circle.

In summary, the MLP classification accuracy is almost same with the SVM classification results, and MLP classifier training data takes much more time tan SVM model. Through figure 3 and figure 5, we can see that they have nearly the same shape of the classification boundary. It is a concentric circle in the plot. Both of the results output by two models are close to maximum accuracy – 85% and 15% probability of error.

Q2

Code for this question can be found in Appendix B

A surfing image was selected from the database.

Preprocessing:
(1) Generate a 5-dimensional feature - row index, column index, red value, green value, blue value for each pixel into a raw feature vector. There are 154401 pixels in total.
(2) Normalize each feature entry individually to the interval [0,1]

The maximum likelihood parameter estimation was used to fit the GMM. The 10-fold cross-validation was used to identify the best GMM model.
After fitting a Gaussian Mixture Model to these normalized feature vectors, the 10-fold cross-validation result is shown below. The horizontal label represents the number of GMMS. The average likelihood for this table was largest in the range from 6 to 10 GMMS.

	1	2	3	4	5	6	7	8	9	10
Log Likelihood (X 10^4)	3.2514	4.5363	5.4276	6.7127	6.3966	7.4935	6.7117	7.8505	7.4651	7.6943

Table 1

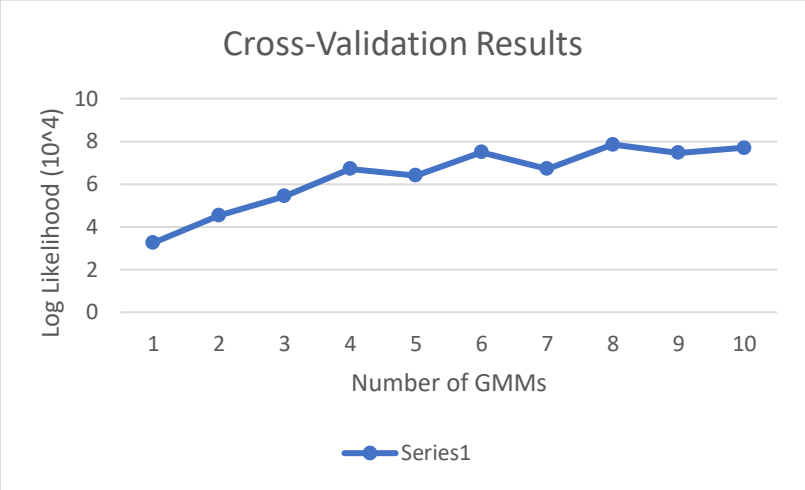

Figure 6

The original image and the GMM-based segmentation outcome are shown below. The segmented image with 2 GMMs is on the medium. The segmented image with the GMMs (8 numbers of GMMs) that maximized the log likelihood is on the right.

Figure 7

Appendix A

```
#!/usr/bin/env python3

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 keras.optimizers import SGD

plotData = True
n = 2
Ntrain = 1000
Ntest = 10000
ClassPriors = [0.35, 0.65]
r0 = 2
r1 = 4
sigma = 1
K = 10

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.title('TestingData')
    plt.show()

# Uses K-Fold cross validation to find the best hyperparameters for an SVM model
# 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 Results")
    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 Results")
    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 Results')
    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()
```

```
(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 B

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%
%EECE5644 Spring 2022
%Homework #4
%Problem #2
%Significant parts of this code were derived from the following sources
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%
if 1 %Set to 0 to just plot existing data
clear all;
close all;
warning off;

%Input data
filenames = {'surf.jpg'};
dTypes={'surfing'};

%Cross validation parameters
NumGMMtoCheck=10;
k=10;
end

for ind=1:length(filenames)
   imdata = imread(filenames{ind});
   figure(1);
   subplot(length(filenames),3,(ind-1)*3+1);
   imshow(imdata);

   if 0 %Skip pre-processing and GMM=2 evaluation
   %Flatten into num features x number of normalized points
   [R,C,D]=size(imdata);
   N=R*C;
```

```
imdata=double(imdata);

rows=(1:R)'*ones(1,C);
columns=ones(R,1)*(1:C);
featureData=[rows(:)';columns(:)'];

for ind2=1:D
    imdatad=imdata(:,:,ind2);
    featureData=[featureData; imdatad(:)'];
end

minf=min(featureData,[],2);
maxf=max(featureData,[],2);
ranges=maxf-minf;

%Normalization
x=(featureData-minf)./ranges;

%Assess GMM with 2 Gaussians case
GMM2=fitgmdist(x',2,'Replicates',10);
post2=posterior(GMM2,x')';
decisions=post2(2,:)>post2(1,:);
end

%Plot GMM=2 case
labelImage2=reshape(decisions,R,C);
subplot(length(filenames),3,(ind-1)*3+2);
imshow(uint8(labelImage2*255/2));

if 0 %Skip k-fold evaluation and training
%Perform k-fold cross-validation to determine optimal number of Gaussians for GMM model
N=length(x);
numValIters=10;
```

```
%Setup cross validation on training data
partSize=floor(N/k);
partInd=[1:partSize:N length(x)];

%Perform cross validation
% select number of perceptrons
for NumGMMs=1:NumGMMtoCheck
   for NumKs=1:k
      index.val=partInd(NumKs):partInd(NumKs+1);
      index.train=setdiff(1:N,index.val);
      %Create object with M perceptrons in hidden layer
      GMMk_loop=fitgmdist(x(:,index.train)',NumGMMs, 'Replicates',5);

      if GMMk_loop.Converged
         probX(NumKs)=sum(log(pdf(GMMk_loop,x(:,index.val)')));
      else
         probX(NumKs)=0;
      end
   end
   %Determine average probability of error
   avgProb(ind,NumGMMs)=mean(probX);
   stats(ind).NumGMMs=1:NumGMMtoCheck;
   stats(ind).avgProb=avgProb;
   stats(ind).mProb(:,NumGMMs)=probX;
   fprintf('File: %1.0f, NumGMM: %1.0f\n',ind,NumGMMs);
end
end

%Select GMM with maximum probability
[~,optNumGMM]=max(avgProb(ind,:));

%optNumGMM=3;
```

```
    GMMk=fitgmdist(x',optNumGMM,'Replicates',10);
    postk=posterior(GMMk,x')';
    lossMatrix=ones(optNumGMM,optNumGMM)-eye(optNumGMM);

    % Expected Risk for each label (rows) of each sample (columns)
    expectedRisks =lossMatrix*postk;

    % Minimum expected risk decision with 0-1 loss
    % Same as MAP
    [~,decisions] = min(expectedRisks,[],1);

    %Plot segmented image for Max. Likelihood number of GMMs case
    labelImageK=reshape(decisions-1,R,C);
    subplot(length(filenames),3,(ind-1)*3+3);
    imshow(uint8(labelImageK*255/optNumGMM));

    save(['HW4_Q2' num2str(ind) '.mat']);
end
```