GE 461 Introduction to Data Science
Project 2 Report
Oğuz Altan - 21600966

In this project, we work on data dimension analysis and reduction, and visualisation of data on high dimensions. We have dataset of 5000 digit image samples, around 500 samples for each class. Each sample is a digitized 20 x 20 image of a digit. As a preaparation, we first divide the data into two subsets by randomly selecting half of the patterns from each class for training and the remaining patterns for testing.

Question 1

We use Principal Component Analysis (PCA) in this part, to project our 400 dimensional image data onto lower dimensional subspaces to observe the effect of dimensionality on the performance of Gaussian classifier.

Using PCA, we first obtain a new set of bases, using training dataset. As a result of decomposition, the eigenvalues in descending order are plotted below:

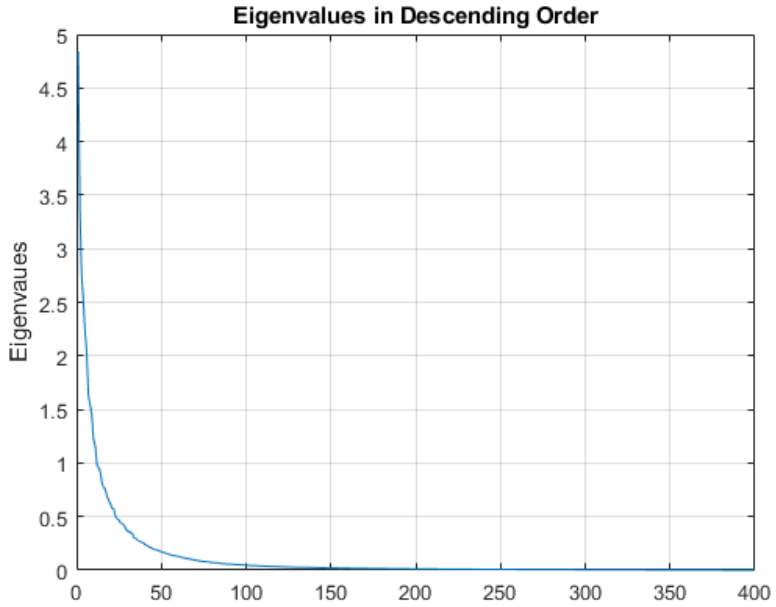

Figure 1: Eigenvalues found by PCA

Looking at the plot, we can see that after the 100th eigenvalue, the values of the rest of the eigenvalues are nearly zero. Therefore, it is a good choice to choose top 100 eigenvalues and coresponding eigenvectors as our bases.

Another way to see that this choice is valid, we can sum up the percentage variances associated with top 100 eigenvalues, as higher variance means higher information and higher importance for that term.

From MATLAB, sum of the total percentage variation is calculated as 93.492%, meaning that 93.492% of the total variance in all eigenvalues are contributed by top 100 eigenvalues. Therefore, top 100 is a good choice.

Sample mean for the whole training data set as an image, using samples for all classes together, is given below. This image contains an average of all digit images in the dataset.

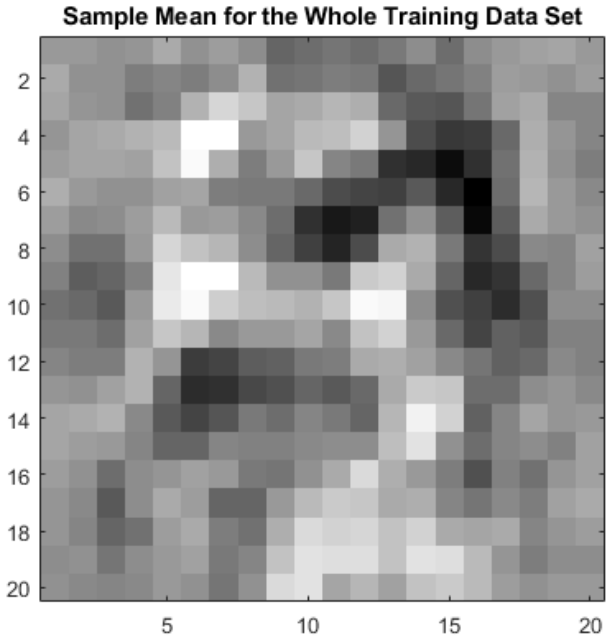

Figure 2: Sample Mean Image

The images of 100 bases (eigenvectors) are given on the next page, for the sake of clarity and ease to observe. Commenting on the bases, we can see that the image get distorted and becomes more different than the original images, as we go from first base to the last. PCA is optimized for reconstruction, so eigenvectors are used for reconstructing the original images. As higher eigenvalue means higher variance and therefore higher information, this result is expected.

Figure 3: Images of Top 100 Bases in Descending Order

Gaussian Classification:

We choose 21 different dimensions, which are 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190 and 200. Then, we project our training and testing datasets using the transformation matrix found using PCA.

To train a Gaussian classifier, we need to first fit a Gaussian denisty for each class. We use the equations for maximum likelihood estimates of the mean vector and covariance matrix:

$$\hat{\boldsymbol{\mu}} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x_i}$$

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x_i}-\hat{\boldsymbol{\mu}})(\mathbf{x_i}-\hat{\boldsymbol{\mu}})^T$$

where n denotes the number of samples in that class and x_i denotes samples. After calculating this parameters, we can calculate the probability of the test sample to be assigned to class c^* using multivariate Gaussian distribution function:

$$p(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}}\exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right]$$

During the classification process, a sample can be assigned to class c^*that gives the highest probability for that sample's feature vector x as

$$c^* = \underset{c=1,\ldots,C}{\arg\max}\; p_c(\mathbf{x}|\boldsymbol{\mu_c},\boldsymbol{\Sigma_c}).$$

After assigning all of the samples in the dataset to a class using this Gaussian classifier, we can compute the succes and also error rate of our classifier by comparing the output of the classifier and ground truth labels. For our system, the success and error rates for both test and train dataset are given below:

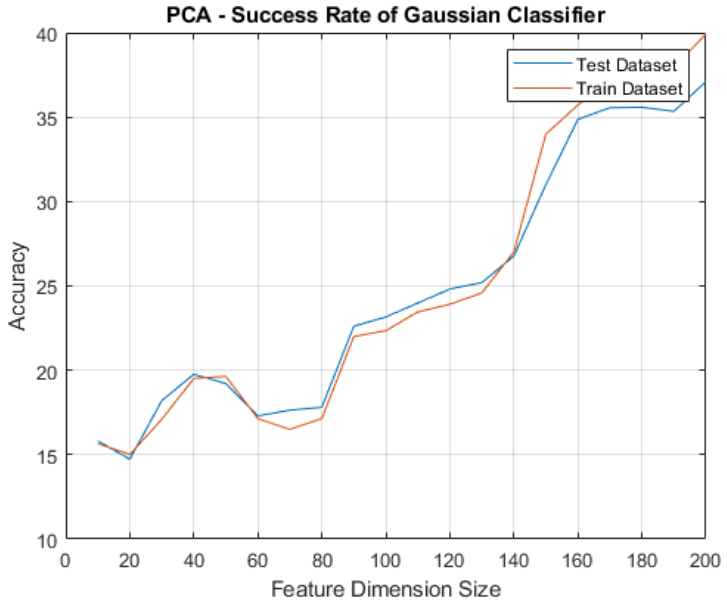

Figure 4: Success Rate of Gaussian Classifier using PCA

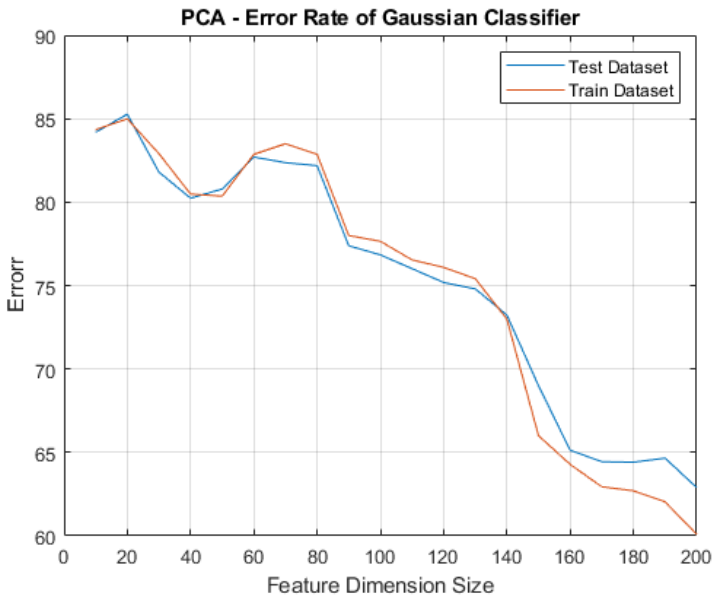

Figure 5: Error Rate of Gaussian Classifier using PCA

We can see that our classifier gets better for higher dimensionality. As the feature dimension size increases, success rate increases and error rate diminishes. This result is expected because work with more dimensions means working with higher variance, therefore we can use more information about the original image dataset and system. We can conclude that our classifier is indeed a successful implementation.

Question 2

We use Fischer linear Discriminant Analysis (LDA) in this part, to again project our 400 dimensional image data onto lower dimensional subspaces.

Using LDA, we obtain new set of bases. This analysis can be done as it follows:

As a first step, we compute within-class scatter matrix S_w:

$$\mathbf{S}_i = \sum_{\mathbf{x}\in\mathcal{D}_i} (\mathbf{x}-\mathbf{m}_i)(\mathbf{x}-\mathbf{m}_i)^T \text{ where } \mathbf{m}_i = \frac{1}{\#\mathcal{D}_i}\sum_{\mathbf{x}\in\mathcal{D}_i}\mathbf{x}.$$

$$\mathbf{S}_\mathbf{W} = \sum_{i=1}^{c} \mathbf{S}_i.$$

Next step, we calculate between-class scatter matrix S_B:

$$\mathbf{S}_\mathbf{B} = \sum_{i=1}^{c} (\#\mathcal{D}_i)(\mathbf{m}_i-\mathbf{m})(\mathbf{m}_i-\mathbf{m})^T$$

The Fischer criterion function now becomes

$$J(\mathbf{W}) = \frac{|\mathbf{W}^T\mathbf{S}_\mathbf{B}\mathbf{W}|}{|\mathbf{W}^T\mathbf{S}_\mathbf{W}\mathbf{W}|}$$

This function is maximized for columns of matrix W to be the eigenvectors of $S_W^{-1}S_B$ having the larges eigenvalues.

The matrix $S_W^{-1}S_B$ has rank of c-1, the number of linear independent eigenvectors, where c is the number of classes. Therefore, we can have at most c-1 nonzero eigenvalues and corresponding c-1 eigenvectors.

In our case, we have 10 classes. Therefore, we can have at most 9 bases (eigenvectors) using LDA on training dataset. We display these 9 bases as images:

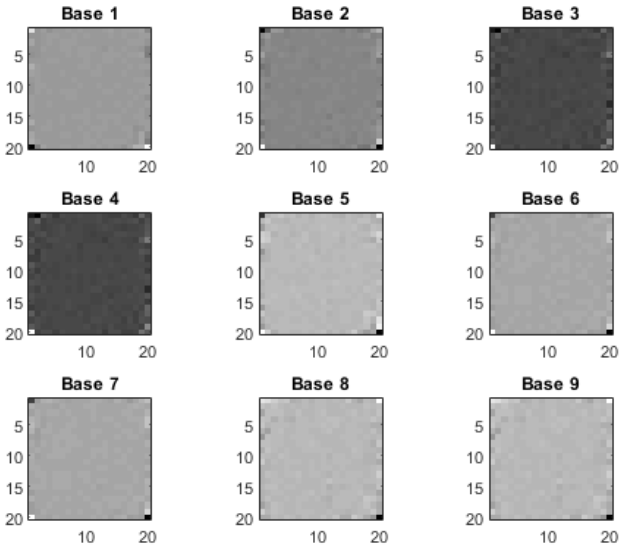

Figure 6: All Bases by LDA as Images

PCA is optimized for reconstruction, whereas LDA is optimized for separation. Therefore, we do not expect bases to look like original data images, but rather we expect them to be a good separation of samples and classes. Therefore, these images of bases are expected results.

Again, as we did in the first question, we train a Gaussian classifier and work on it.

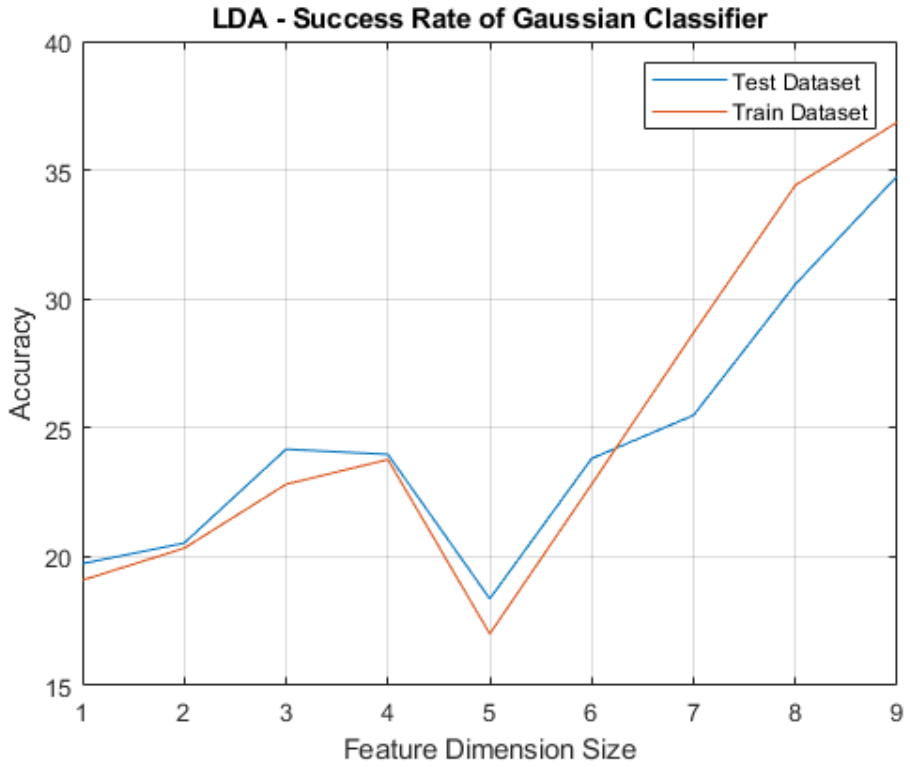

Figure 7: Success Rate of Gaussian Classifier using LDA

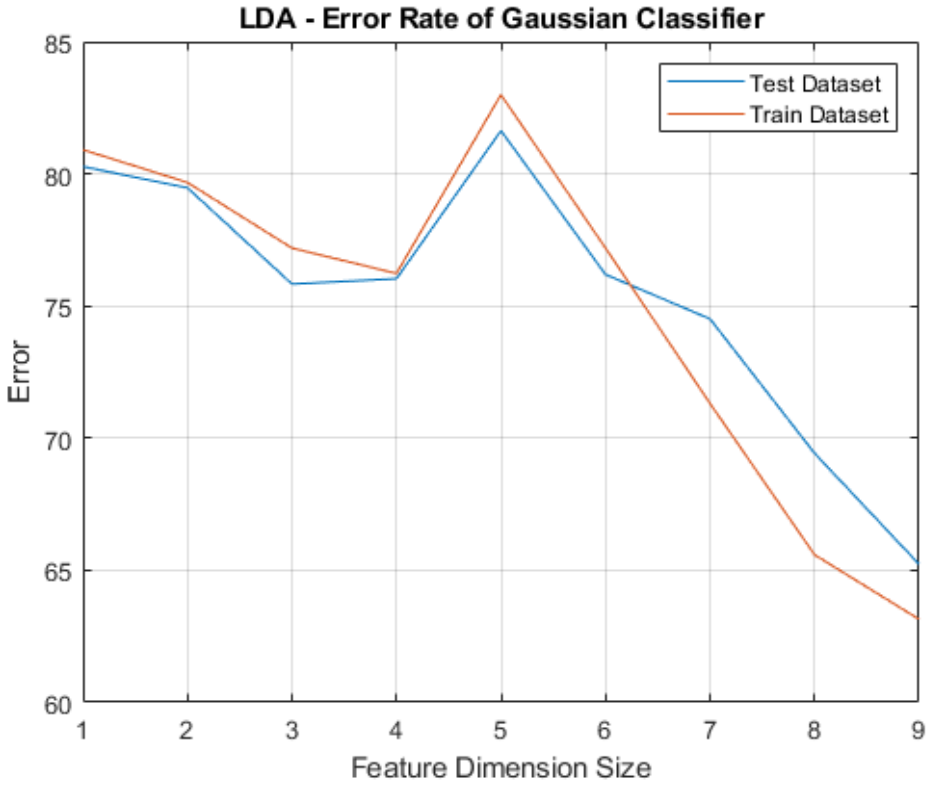

Figure 8: Error Rate of Gaussian Classifier using LDA

We can see that our classifier gets better for higher dimensionality. As the feature dimension size increases, success rate increases and error rate diminishes. This result is expected because work with more dimensions means working with higher variance, therefore we can use more information about the original image dataset and system. We can conclude that our classifier is indeed a successful implementation.

Comparing this method,LDA, with PCA, we can see that there is not any clear winner, so both methods are as good as each other in our case and for our dataset.

Question 3

In this question, we use two methods for dimensionality reduction particularly designed for visualisation high-dimensionla data sets. First one is Sammon's mapping and second one is t-SNe. Both of these methods are used for mapping the digit data set to two dimensions.

Using Sammon's mapping with default options, the resulting mapping for the whole data set and scatter of the patterns together with their class information are given on the next page.

This result is for the default setting of Sammon.m MATLAB function, where the maximum iteration MaxIter of this iterative approach is 500 iterations and the stopping criteria TolFun is 10^{-9} , meaning that if the difference of errors between two consecutive iterations becomes lower than this criteria 10^{-9}, then process stops as program assumes that result is optimized.

Increasing this maximum iterations to 2000 and experimenting on stopping criteria increase the separation and classification performance better. The result is on the page after the next one.

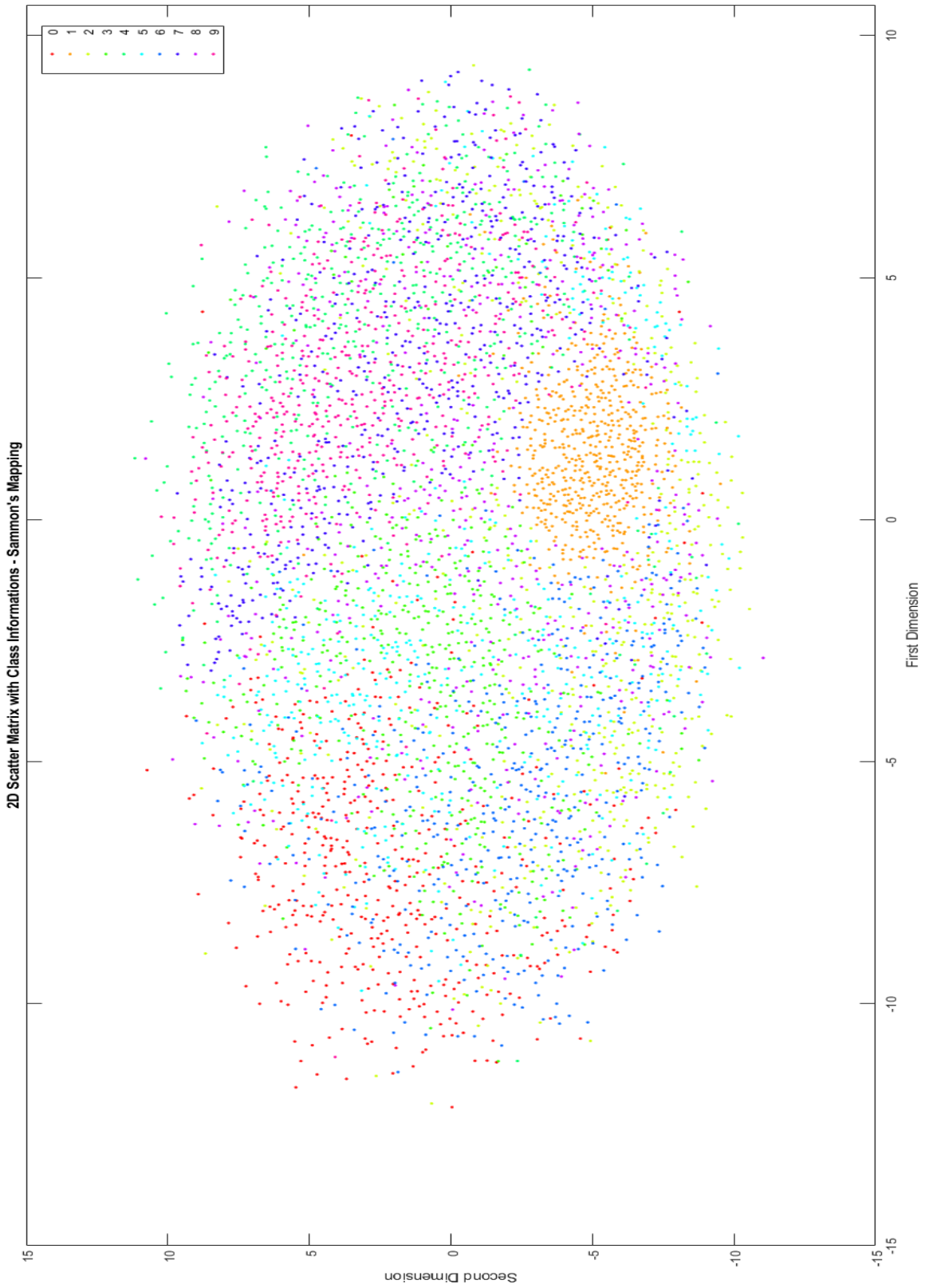

Figure 9: Sammon's Mapping with Default Options

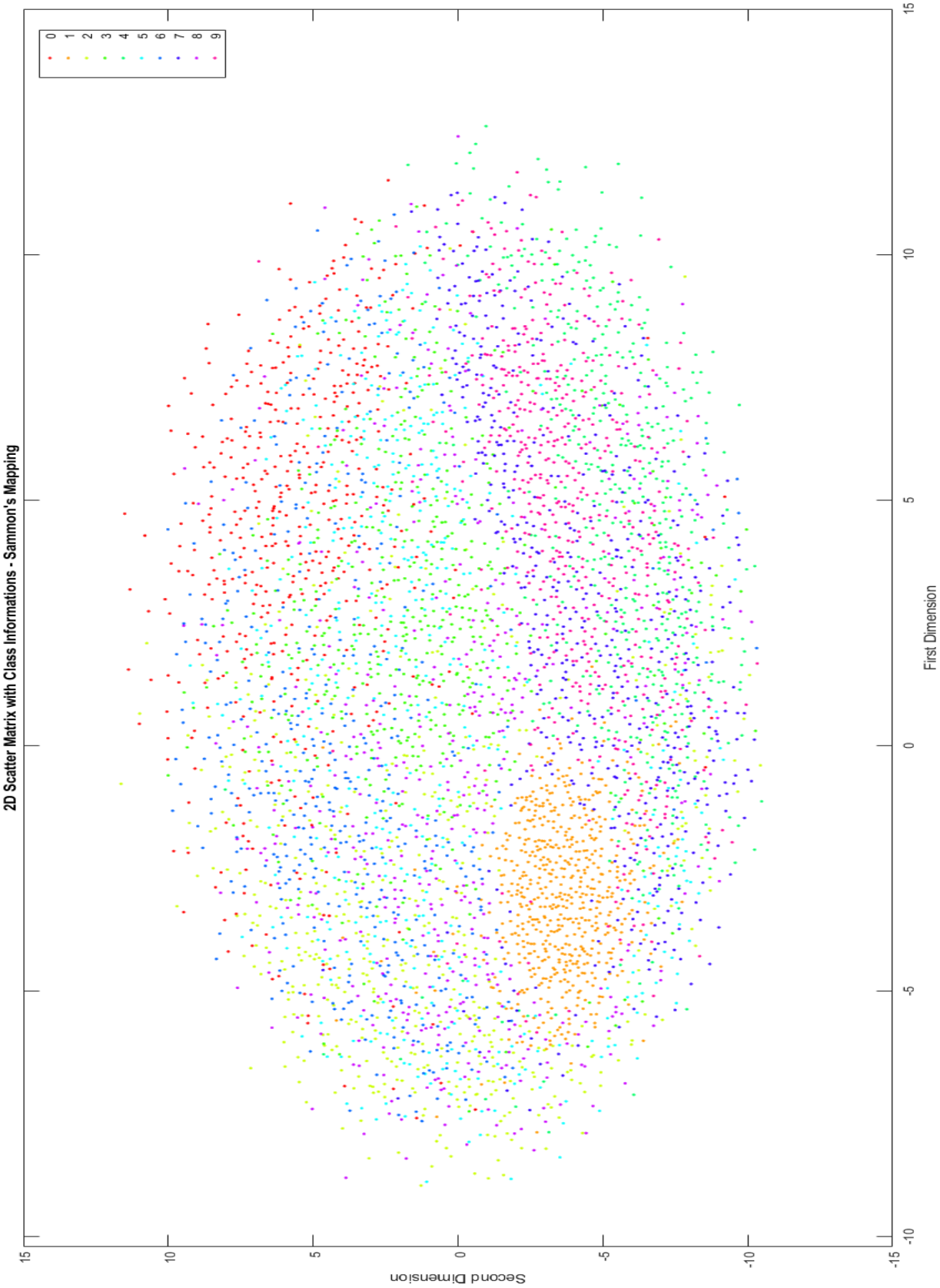

Figure 10: Sammon's Mapping with MaxIter = 2000 and TolFun = 10^{-8}

Using two plots, it can be said that both of the mappings are successful for visualisation of high dimension data, and the second mapping with improved parameters can separate data better than mapping with default options.

Now, we use t-SNE (t-Distributed Stochastic Neighbor Embedding) for the mapping the digit data set to two dimensions. Using tSNE with default options, the resulting mapping for the whole data set and scatter of the patterns together with their class information are given:

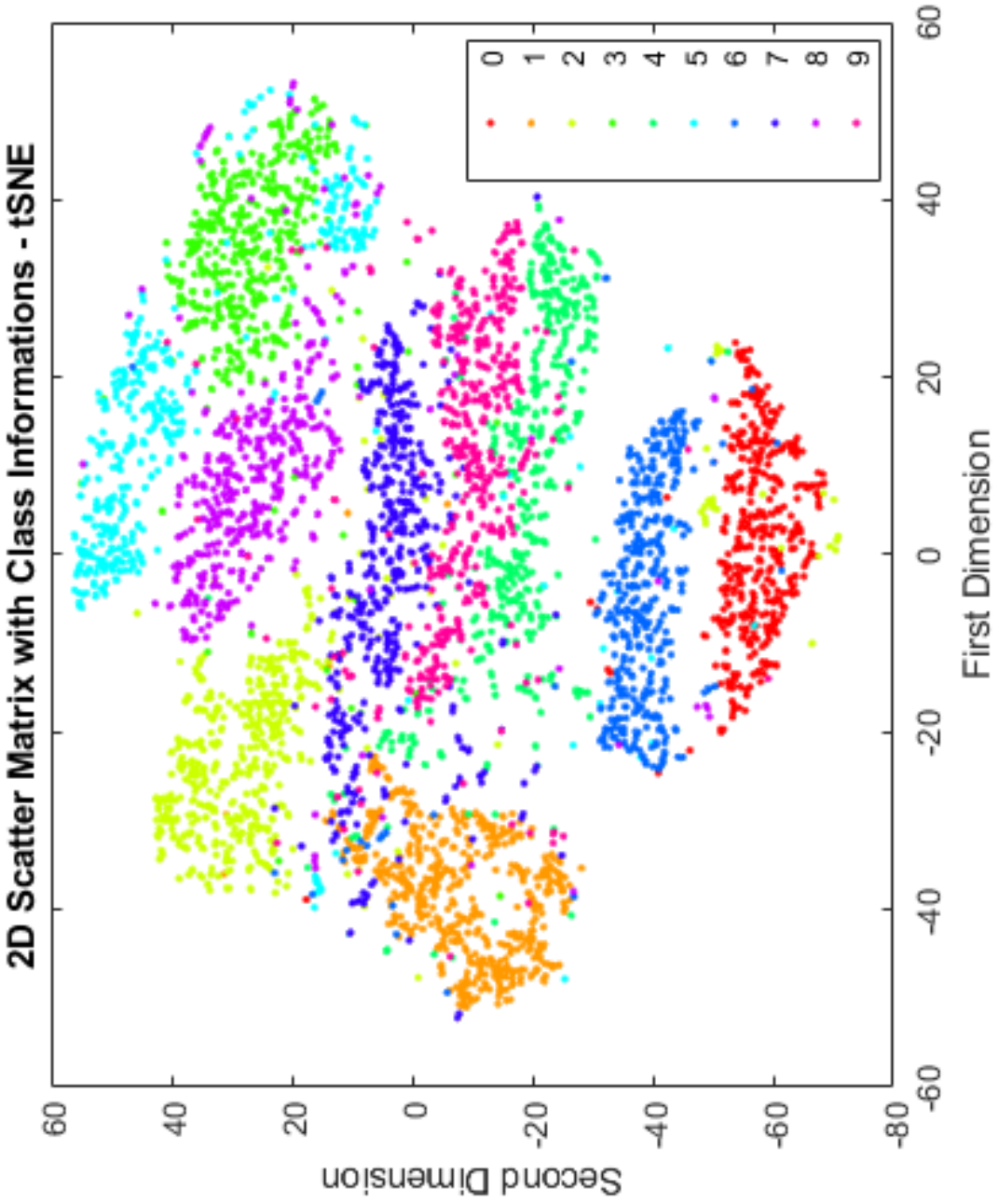

Figure 11: t-SNE Visualization of High-Dimension Data

We can see that t-SNE can separate the data to the releveant classes and perform visualisation of high-dimension data successfuly.

Appendix

A) References

1. Sammon, John W. Jr., "A Nonlinear Mapping for Data Structure Analysis", IEEE Transactions on Computers, vol. C-18, no. 5,pp 401-409, May 1969.

2. Cawley, Gavin C. And Talbot, Nicola L. (2011) sammon.m (Version 1.20) [Source code]

https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/56937/versions/28/previews/FSLib_v6.0_2018/lib/drtoolbox/techniques/sammon.m/index.html

B) Code

pca.m

```
%%
clear all; clc;
close all;

load digits.mat;

% data is already between 0 and 1
digits = digits - mean(digits); %normalizing data
% digits = digits/255;

labelCount = [];
for i = 0:9
    labelc = labels == i;
    labelCount = [labelCount sum(labelc)];
end

% sorting data and labels in increasing order
[labels,isort] = sort(labels);
digits = digits(isort,:);

trainSet = digits(1:230,:);
testSet = digits(231:460,:);
trainLabel = labels(1:230,:);
testLabel = labels(231:460,:);

trainSet = [trainSet;digits(461:745,:)];
testSet = [testSet;digits(747:1031,:)];
trainLabel = [trainLabel;labels(461:745,:)];
testLabel = [testLabel;labels(747:1031,:)];

trainSet = [trainSet;digits(1032:1296,:)];
testSet = [testSet;digits(1297:1561,:)];
trainLabel = [trainLabel;labels(1032:1296,:)];
testLabel = [testLabel;labels(1297:1561,:)];

trainSet = [trainSet;digits(1562:1811,:)];
testSet = [testSet;digits(1812:2061,:)];
trainLabel = [trainLabel;labels(1562:1811,:)];
testLabel = [testLabel;labels(1812:2061,:)];
```

```
trainSet = [trainSet;digits(2062:2311,:)];
testSet = [testSet;digits(2312:2561,:)];
trainLabel = [trainLabel;labels(2062:2311,:)];
testLabel = [testLabel;labels(2312:2561,:)];

trainSet = [trainSet;digits(2562:2789,:)];
testSet = [testSet;digits(2790:3017,:)];
trainLabel = [trainLabel;labels(2562:2789,:)];
testLabel = [testLabel;labels(2790:3017,:)];

trainSet = [trainSet;digits(3018:3248,:)];
testSet = [testSet;digits(3249:3479,:)];
trainLabel = [trainLabel;labels(3018:3248,:)];
testLabel = [testLabel;labels(3249:3479,:)];

trainSet = [trainSet;digits(3480:3735,:)];
testSet = [testSet;digits(3736:3991,:)];
trainLabel = [trainLabel;labels(3480:3735,:)];
testLabel = [testLabel;labels(3480:3735,:)];

trainSet = [trainSet;digits(3992:4235,:)];
testSet = [testSet;digits(4237:4480,:)];
trainLabel = [trainLabel;labels(3992:4235,:)];
testLabel = [testLabel;labels(3992:4235,:)];

trainSet = [trainSet;digits(4481:4740,:)];
testSet = [testSet;digits(4741:5000,:)];
trainLabel = [trainLabel;labels(4481:4740,:)];
testLabel =  [testLabel;labels(4741:5000,:)];

% PCA code
% coeffs are eigenvectors /principal component vectors. These are the
eigenvectors of the covariance matrix
% scores are the projection of the data in the principal component space
% explained is percentage of variance by each eigenvalue
% coeff = eigenvectors
% score = newDataSet;
[coeff, score, latent, tsquared, explained, mu0] = pca(trainSet); %applying
PCA

% Column vectors of V are eigenvector. Diagonals of D are eigenvalues
covarianceMatrix = cov(trainSet);
[V,D] = eig(covarianceMatrix);
V = fliplr(V);

dataInPrincipalComponentSpace = trainSet*coeff;

% The variances of these vectors are the eigenvalues of the covariance
matrix, and are also the output "latent".
variances = var(dataInPrincipalComponentSpace)';
eigenvalues = latent;
sortedD = sort(diag(D),'descend');

figure;
stem(1:400,latent);
grid on
title("Eigenvalues in Descending Order");
```

```
ylabel("Eigenvaues");

% images for first 10 eigenvectors corresponding to highest 10 eigenvalues
for i = 1:100
    subplot(10,10,i);
    imagesc(reshape(V(:,i), 20, 20));
    colormap(gray);
    axis image;
    title("Eigenvector " + i);
    hold on;
end

% total percentage of variance contributed by top 100 eigenvalues
percentageVariance = sum(explained(1:100,:));

% displaying the sample mean for the whole training data set as an image
A = mean(trainSet);
figure;
imagesc(reshape(A,20, 20));
colormap(gray);
axis image;
title("Sample Mean for the Whole Training Data Set");

% Multiply the original data by the principal component vectors to get the
projections of the original data on the
% principal component vector space. This is also the output "score".

% they are equal if data is normalized
reducDataSet = trainSet*V; %choose 99 dimensions 10*10
X_reduce = score;
% [~, pca_scores, ~, ~, var_explained] = pca(trainSet, 'NumComponents',
10);

correlation_mat = corrcoef(dataInPrincipalComponentSpace);
correlation_mat2 = corrcoef(reducDataSet);
correlation_mat3 = corrcoef(X_reduce);

% 21 different subspace dimension and projected train and testsets
trainDataSet1 = trainSet*V(:,1:1); %dimension 1
trainDataSet10 = trainSet*V(:,1:10); %dimension 10
trainDataSet20 = trainSet*V(:,1:20); %dimension 20
trainDataSet30 = trainSet*V(:,1:30); %dimension 30
trainDataSet40 = trainSet*V(:,1:40); %dimension 40
trainDataSet50 = trainSet*V(:,1:50); %dimension 50
trainDataSet60 = trainSet*V(:,1:60); %dimension 60
trainDataSet70 = trainSet*V(:,1:70); %dimension 70
trainDataSet80 = trainSet*V(:,1:80); %dimension 80
trainDataSet90 = trainSet*V(:,1:90); %dimension 90
trainDataSet100 = trainSet*V(:,1:100); %dimension 100
trainDataSet110 = trainSet*V(:,1:110); %dimension 110
trainDataSet120 = trainSet*V(:,1:120); %dimension 120
trainDataSet130 = trainSet*V(:,1:130); %dimension 130
trainDataSet140 = trainSet*V(:,1:140); %dimension 140
trainDataSet150 = trainSet*V(:,1:150); %dimension 150
trainDataSet160 = trainSet*V(:,1:160); %dimension 160
trainDataSet170 = trainSet*V(:,1:170); %dimension 170
trainDataSet180 = trainSet*V(:,1:180); %dimension 180
trainDataSet190 = trainSet*V(:,1:190); %dimension 190
trainDataSet200 = trainSet*V(:,1:200); %dimension 200
```

```
testDataSet1 = testSet*V(:,1:1); %dimension 1
testDataSet10 = testSet*V(:,1:10); %dimension 10
testDataSet20 = testSet*V(:,1:20); %dimension 20
testDataSet30 = testSet*V(:,1:30); %dimension 30
testDataSet40 = testSet*V(:,1:40); %dimension 40
testDataSet50 = testSet*V(:,1:50); %dimension 50
testDataSet60 = testSet*V(:,1:60); %dimension 60
testDataSet70 = testSet*V(:,1:70); %dimension 70
testDataSet80 = testSet*V(:,1:80); %dimension 80
testDataSet90 = testSet*V(:,1:90); %dimension 90
testDataSet100 = testSet*V(:,1:100); %dimension 100
testDataSet110 = testSet*V(:,1:110); %dimension 110
testDataSet120 = testSet*V(:,1:120); %dimension 120
testDataSet130 = testSet*V(:,1:130); %dimension 130
testDataSet140 = testSet*V(:,1:140); %dimension 140
testDataSet150 = testSet*V(:,1:150); %dimension 150
testDataSet160 = testSet*V(:,1:160); %dimension 160
testDataSet170 = testSet*V(:,1:170); %dimension 170
testDataSet180 = testSet*V(:,1:180); %dimension 180
testDataSet190 = testSet*V(:,1:190); %dimension 190
testDataSet200 = testSet*V(:,1:200); %dimension 200

%% Gaussian Classifier

% figure;
% imagesc(reshape(trainDataSet110(1,:), 10, 10));
% colormap(gray);
% axis image;

% fitting MLE parameters for Gaussian fitting

mu0 = mean(trainDataSet200(1:230,:));
mu1 = mean(trainDataSet200(231:515,:));
mu2 = mean(trainDataSet200(516:780,:));
mu3 = mean(trainDataSet200(781:1030,:));
mu4 = mean(trainDataSet200(1031:1280,:));
mu5 = mean(trainDataSet200(1281:1508,:));
mu6 = mean(trainDataSet200(1509:1739,:));
mu7 = mean(trainDataSet200(1740:1995,:));
mu8 = mean(trainDataSet200(1996:2239,:));
mu9 = mean(trainDataSet200(2240:2499,:));

% class 0
sigma0 = 0;
for i = 1:230
    sigma0 = sigma0 + (trainDataSet200(i,:)'-mu0)*(trainDataSet200(i,:)'-
mu0)';
end
sigma0 = sigma0/230;
% sigma0 = abs(sigma0);

% class 1
sigma1 = 0;
for i = 231:515
    sigma1 = sigma1 + (trainDataSet200(i,:)'-mu1)*(trainDataSet200(i,:)'-
mu1)';
end
sigma1 = sigma1/285;
```

```
% sigma1 = abs(sigma1);

%class 2
sigma2 = 0;
for i = 516:780
    sigma2 = sigma2 + (trainDataSet200(i,:)'-mu2)*(trainDataSet200(i,:)'-
mu2)';
end
sigma2 = sigma2/265;
% sigma2 = abs(sigma2);

%class 3
sigma3 = 0;
for i = 781:1030
    sigma3 = sigma3 + (trainDataSet200(i,:)'-mu3)*(trainDataSet200(i,:)'-
mu3)';
end
sigma3 = sigma3/250;
% sigma3 = abs(sigma3);

%class 4
sigma4 = 0;
for i = 1031:1280
    sigma4 = sigma4 + (trainDataSet200(i,:)'-mu4)*(trainDataSet200(i,:)'-
mu4)';
end
sigma4 = sigma4/250;
% sigma4 = abs(sigma4);

%class 5
sigma5 = 0;
for i = 1281:1508
    sigma5 = sigma5 + (trainDataSet200(i,:)'-mu5)*(trainDataSet200(i,:)'-
mu5)';
end
sigma5 = sigma5/228;
% sigma5 = abs(sigma5);

%class 6
sigma6 = 0;
for i = 1509:1739
    sigma6 = sigma6 + (trainDataSet200(i,:)'-mu6)*(trainDataSet200(i,:)'-
mu6)';
end
sigma6 = sigma6/256;
% sigma6 = abs(sigma6);

%class 7
sigma7 = 0;
for i = 1740:1995
    sigma7 = sigma7 + (trainDataSet200(i,:)'-mu7)*(trainDataSet200(i,:)'-
mu7)';
end
sigma7 = sigma7/256;
% sigma7 = abs(sigma7);

%class 8
sigma8 = 0;
for i = 1996:2239
```

```
    sigma8 = sigma8 + (trainDataSet200(i,:)'-mu8)*(trainDataSet200(i,:)'-
mu8)';
end
sigma8 = sigma8/244;
% sigma8 = abs(sigma8);

%class 9
sigma9 = 0;
for i = 2240:2499
    sigma9 = sigma9 + (trainDataSet200(i,:)'-mu9)*(trainDataSet200(i,:)'-
mu9)';
end
sigma9 = sigma9/260;
% sigma9 = abs(sigma9);

% testSet classification
test_class = [];

for i = 1:2499
    x = testDataSet200(i,:);
    y0 = 1/((((2*pi)^200)*det(sigma0))^1/2)*exp(-0.5*(x-
mu0)*inv(sigma0)*(x-mu0'));
    y1 = 1/((((2*pi)^200)*det(sigma1))^1/2)*exp(-0.5*(x-
mu1)*inv(sigma1)*(x-mu1)');
    y2 = 1/((((2*pi)^200)*det(sigma2))^1/2)*exp(-0.5*(x-
mu2)*inv(sigma2)*(x-mu2)');
    y3 = 1/((((2*pi)^200)*det(sigma3))^1/2)*exp(-0.5*(x-
mu3)*inv(sigma3)*(x-mu3)');
    y4 = 1/((((2*pi)^200)*det(sigma4))^1/2)*exp(-0.5*(x-
mu4)*inv(sigma4)*(x-mu4)');
    y5 = 1/((((2*pi)^200)*det(sigma5))^1/2)*exp(-0.5*(x-
mu5)*inv(sigma5)*(x-mu5)');
    y6 = 1/((((2*pi)^200)*det(sigma6))^1/2)*exp(-0.5*(x-
mu6)*inv(sigma6)*(x-mu6)');
    y7 = 1/((((2*pi)^200)*det(sigma7))^1/2)*exp(-0.5*(x-
mu7)*inv(sigma7)*(x-mu7)');
    y8 = 1/((((2*pi)^200)*det(sigma8))^1/2)*exp(-0.5*(x-
mu8)*inv(sigma8)*(x-mu8)');
    y9 = 1/((((2*pi)^200)*det(sigma9))^1/2)*exp(-0.5*(x-
mu9)*inv(sigma9)*(x-mu9)');

    y_test_vec = [y0 y1 y2 y3 y4 y5 y6 y7 y8 y9];
    [maxx1,test_ind] = max(y_test_vec);
    test_class = [test_class test_ind-1];
end

test_correct = sum(test_class == testLabel')/2499*100

% trainSet classification
train_class = [];

for j = 1:2499
    x = trainDataSet200(j,:);
    y0 = 1/((((2*pi)^200)*det(sigma0))^1/2)*exp(-0.5*(x-
mu0)*inv(sigma0)*(x-mu0)');
    y1 = 1/((((2*pi)^200)*det(sigma1))^1/2)*exp(-0.5*(x-
mu1)*inv(sigma1)*(x-mu1)');
    y2 = 1/((((2*pi)^200)*det(sigma2))^1/2)*exp(-0.5*(x-
mu2)*inv(sigma2)*(x-mu2)');
```

```
    y3 = 1/((((2*pi)^200)*det(sigma3))^1/2)*exp(-0.5*(x-
mu3)*inv(sigma3)*(x-mu3)');
    y4 = 1/((((2*pi)^200)*det(sigma4))^1/2)*exp(-0.5*(x-
mu4)*inv(sigma4)*(x-mu4)');
    y5 = 1/((((2*pi)^200)*det(sigma5))^1/2)*exp(-0.5*(x-
mu5)*inv(sigma5)*(x-mu5)');
    y6 = 1/((((2*pi)^200)*det(sigma6))^1/2)*exp(-0.5*(x-
mu6)*inv(sigma6)*(x-mu6)');
    y7 = 1/((((2*pi)^200)*det(sigma7))^1/2)*exp(-0.5*(x-
mu7)*inv(sigma7)*(x-mu7)');
    y8 = 1/((((2*pi)^200)*det(sigma8))^1/2)*exp(-0.5*(x-
mu8)*inv(sigma8)*(x-mu8)');
    y9 = 1/((((2*pi)^200)*det(sigma9))^1/2)*exp(-0.5*(x-
mu9)*inv(sigma9)*(x-mu9)');

    y_train_vec = [y0 y1 y2 y3 y4 y5 y6 y7 y8 y9];
    [maxx2,train_ind] = max(y_train_vec);
    train_class = [train_class train_ind-1];
end

train_correct = sum(train_class == trainLabel')/2499*100

test_correct_vec = [15.8156   14.7320  18.2031   19.7652   19.2136
17.2989   17.6373   17.8163   22.6146   23.1626...
23.9867   24.8165   25.1991   26.7692  31.0053   34.8694   35.5591
35.5858   35.3443  37.1057];
train_correct_vec = [15.6602   15.0104   17.0996   19.5196   19.6416
17.1391   16.5019   17.1357   22.0025   22.3490...
23.4615   23.9105   24.5881   27.0150   33.9953 35.7186   37.0585   37.2990
37.9598   39.9160];
dimension_vec = [10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170
180 190 200];

figure;
plot(dimension_vec,test_correct_vec);
grid on
xlabel("Feature Dimension Size");
ylabel("Accuracy");
hold on
plot(dimension_vec,train_correct_vec);
grid on
title("PCA - Success Rate of Gaussian Classifier");
xlabel("Feature Dimension Size");
ylabel("Accuracy");
legend("Test Dataset","Train Dataset");

figure;
plot(dimension_vec,100-test_correct_vec);
grid on
xlabel("Feature Dimension Size");
ylabel("Accuracy");
hold on
plot(dimension_vec,100-train_correct_vec);
grid on
title("PCA - Error Rate of Gaussian Classifier");
xlabel("Feature Dimension Size");
ylabel("Error");
legend("Test Dataset","Train Dataset");
```

lda.m

```
clear all; clc;
close all;

load digits.mat;

% data is already between 0 and 1
digits = digits - mean(digits); %normalizing data
% digits = digits/255;

labelCount = [];
for i = 0:9
    labelc = labels == i;
    labelCount = [labelCount sum(labelc)];
end

% sorting data and labels in increasing order
[labels,isort] = sort(labels);
digits = digits(isort,:);

trainSet = digits(1:230,:);
testSet = digits(231:460,:);
trainLabel = labels(1:230,:);
testLabel = labels(231:460,:);

trainSet = [trainSet;digits(461:745,:)];
testSet = [testSet;digits(747:1031,:)];
trainLabel = [trainLabel;labels(461:745,:)];
testLabel = [testLabel;labels(747:1031,:)];

trainSet = [trainSet;digits(1032:1296,:)];
testSet = [testSet;digits(1297:1561,:)];
trainLabel = [trainLabel;labels(1032:1296,:)];
testLabel = [testLabel;labels(1297:1561,:)];

trainSet = [trainSet;digits(1562:1811,:)];
testSet = [testSet;digits(1812:2061,:)];
trainLabel = [trainLabel;labels(1562:1811,:)];
testLabel = [testLabel;labels(1812:2061,:)];

trainSet = [trainSet;digits(2062:2311,:)];
testSet = [testSet;digits(2312:2561,:)];
trainLabel = [trainLabel;labels(2062:2311,:)];
testLabel = [testLabel;labels(2312:2561,:)];

trainSet = [trainSet;digits(2562:2789,:)];
testSet = [testSet;digits(2790:3017,:)];
trainLabel = [trainLabel;labels(2562:2789,:)];
testLabel = [testLabel;labels(2790:3017,:)];

trainSet = [trainSet;digits(3018:3248,:)];
testSet = [testSet;digits(3249:3479,:)];
trainLabel = [trainLabel;labels(3018:3248,:)];
testLabel = [testLabel;labels(3249:3479,:)];

trainSet = [trainSet;digits(3480:3735,:)];
testSet = [testSet;digits(3736:3991,:)];
```

```
trainLabel = [trainLabel;labels(3480:3735,:)];
testLabel = [testLabel;labels(3480:3735,:)];

trainSet = [trainSet;digits(3992:4235,:)];
testSet = [testSet;digits(4237:4480,:)];
trainLabel = [trainLabel;labels(3992:4235,:)];
testLabel = [testLabel;labels(3992:4235,:)];

trainSet = [trainSet;digits(4481:4740,:)];
testSet = [testSet;digits(4741:5000,:)];
trainLabel = [trainLabel;labels(4481:4740,:)];
testLabel =  [testLabel;labels(4741:5000,:)];

% LDA Code

mu0 = mean(trainSet(1:230,:)); %class 0
mu1 = mean(trainSet(231:515,:)); %class 1
mu2 = mean(trainSet(516:780,:)); %class 2
mu3 = mean(trainSet(781:1030,:)); %class 3
mu4 = mean(trainSet(1031:1280,:)); %class 4
mu5 = mean(trainSet(1281:1508,:)); %class 5
mu6 = mean(trainSet(1509:1739,:)); %class 6
mu7 = mean(trainSet(1740:1995,:)); %class 7
mu8 = mean(trainSet(1996:2239,:)); %class 8
mu9 = mean(trainSet(2240:2499,:)); %class 9

sw0 = 0;
for i = 1:230
    sw0 = sw0+(trainSet(i,:)-mu0)'*(trainSet(i,:)-mu0);
end

sw1 = 0;
for i = 231:515
    sw1 = sw1+(trainSet(i,:)-mu1)'*(trainSet(i,:)-mu1);
end

sw2 = 0;
for i = 516:780
    sw2 = sw2+(trainSet(i,:)-mu2)'*(trainSet(i,:)-mu2);
end

sw3 = 0;
for i = 781:1030
    sw3 = sw3+(trainSet(i,:)-mu3)'*(trainSet(i,:)-mu3);
end

sw4 = 0;
for i = 1031:1280
    sw4 = sw4+(trainSet(i,:)-mu4)'*(trainSet(i,:)-mu4);
end

sw5 = 0;
for i = 1281:1508
    sw5 = sw5+(trainSet(i,:)-mu5)'*(trainSet(i,:)-mu5);
end

sw6 = 0;
for i = 1509:1739
```

```
    sw6 = sw6+(trainSet(i,:)-mu6)'*(trainSet(i,:)-mu6);
end

sw7 = 0;
for i = 1740:1995
    sw7 = sw7+(trainSet(i,:)-mu7)'*(trainSet(i,:)-mu7);
end

sw8 = 0;
for i = 1996:2239
    sw8 = sw8+(trainSet(i,:)-mu8)'*(trainSet(i,:)-mu8);
end

sw9 = 0;
for i = 2240:2499
    sw9 = sw9+(trainSet(i,:)-mu9)'*(trainSet(i,:)-mu0);
end

sw = sw0+sw1+sw2+sw3+sw4+sw5+4+sw6+sw7+sw8+sw9;

mu_all = mean(trainSet);

sb0 = 230.*(mu0-mu_all)'*(mu0-mu_all);
sb1 = 285.*(mu1-mu_all)'*(mu1-mu_all);
sb2 = 265.*(mu2-mu_all)'*(mu2-mu_all);
sb3 = 250.*(mu3-mu_all)'*(mu3-mu_all);
sb4 = 250.*(mu4-mu_all)'*(mu4-mu_all);
sb5 = 228.*(mu5-mu_all)'*(mu5-mu_all);
sb6 = 231.*(mu6-mu_all)'*(mu6-mu_all);
sb7 = 256.*(mu7-mu_all)'*(mu7-mu_all);
sb8 = 244.*(mu8-mu_all)'*(mu8-mu_all);
sb9 = 260.*(mu9-mu_all)'*(mu9-mu_all);

sb = sb0+sb1+sb2+sb3+sb4+sb5+sb6+sb7+sb8+sb9;

proj_matrix= inv(sw)*sb;
rank = rank(proj_matrix); %rank of this matrix is c-1 = 10-1 = 9
[V_lda,D_lda] = eig(proj_matrix);
V_lda = real(fliplr(V_lda));

% images of 9 bases
for i = 1:9
    subplot(3,3,i);
    imagesc(reshape(real(V_lda(:,i)), 20, 20));
    subplot(3,3,i);
    colormap(gray);
    axis image;
    title("Base " + i);
    hold on;
end

%% Gaussian Classification

%projection matrix
W1 = V_lda(:,1);
W2 = V_lda(:,1:2);
W3 = V_lda(:,1:3);
W4 = V_lda(:,1:4);
```

```
W5 = V_lda(:,1:5);
W6 = V_lda(:,1:6);
W7 = V_lda(:,1:7);
W8 = V_lda(:,1:8);
W9 = V_lda(:,1:9);

% 9 different subspace dimension and projected train and testsets
trainDataSet1 = trainSet*W1; %dimension 1
trainDataSet2 = trainSet*W2; %dimension 2
trainDataSet3 = trainSet*W3; %dimension 3
trainDataSet4 = trainSet*W4; %dimension 4
trainDataSet5 = trainSet*W5; %dimension 5
trainDataSet6 = trainSet*W6; %dimension 6
trainDataSet7 = trainSet*W7; %dimension 7
trainDataSet8 = trainSet*W8; %dimension 8
trainDataSet9 = trainSet*W9; %dimension 0

testDataSet1 = testSet*W1; %dimension 1
testDataSet2 = testSet*W2; %dimension 2
testDataSet3 = testSet*W3; %dimension 3
testDataSet4 = testSet*W4; %dimension 4
testDataSet5 = testSet*W5; %dimension 5
testDataSet6 = testSet*W6; %dimension 6
testDataSet7 = testSet*W7; %dimension 7
testDataSet8 = testSet*W8; %dimension 8
testDataSet9 = testSet*W9; %dimension 9

mu0 = mean(trainDataSet9(1:230,:));
mu1 = mean(trainDataSet9(231:515,:));
mu2 = mean(trainDataSet9(516:780,:));
mu3 = mean(trainDataSet9(781:1030,:));
mu4 = mean(trainDataSet9(1031:1280,:));
mu5 = mean(trainDataSet9(1281:1508,:));
mu6 = mean(trainDataSet9(1509:1739,:));
mu7 = mean(trainDataSet9(1740:1995,:));
mu8 = mean(trainDataSet9(1996:2239,:));
mu9 = mean(trainDataSet9(2240:2499,:));

% class 0
sigma0 = 0;
for i = 1:230
    sigma0 = sigma0 + (trainDataSet9(i,:)'-mu0)*(trainDataSet9(i,:)'-mu0)';
end
sigma0 = sigma0/230;
% sigma0 = abs(sigma0);

% class 1
sigma1 = 0;
for i = 231:515
    sigma1 = sigma1 + (trainDataSet9(i,:)'-mu1)*(trainDataSet9(i,:)'-mu1)';
end
sigma1 = sigma1/285;
% sigma1 = abs(sigma1);

%class 2
sigma2 = 0;
for i = 516:780
    sigma2 = sigma2 + (trainDataSet9(i,:)'-mu2)*(trainDataSet9(i,:)'-mu2)';
end
```

```
sigma2 = sigma2/265;
% sigma2 = abs(sigma2);

%class 3
sigma3 = 0;
for i = 781:1030
    sigma3 = sigma3 + (trainDataSet9(i,:)'-mu3)*(trainDataSet9(i,:)'-mu3)';
end
sigma3 = sigma3/250;
% sigma3 = abs(sigma3);

%class 4
sigma4 = 0;
for i = 1031:1280
    sigma4 = sigma4 + (trainDataSet9(i,:)'-mu4)*(trainDataSet9(i,:)'-mu4)';
end
sigma4 = sigma4/250;
% sigma4 = abs(sigma4);

%class 5
sigma5 = 0;
for i = 1281:1508
    sigma5 = sigma5 + (trainDataSet9(i,:)'-mu5)*(trainDataSet9(i,:)'-mu5)';
end
sigma5 = sigma5/228;
% sigma5 = abs(sigma5);

%class 6
sigma6 = 0;
for i = 1509:1739
    sigma6 = sigma6 + (trainDataSet9(i,:)'-mu6)*(trainDataSet9(i,:)'-mu6)';
end
sigma6 = sigma6/256;
% sigma6 = abs(sigma6);

%class 7
sigma7 = 0;
for i = 1740:1995
    sigma7 = sigma7 + (trainDataSet9(i,:)'-mu7)*(trainDataSet9(i,:)'-mu7)';
end
sigma7 = sigma7/256;
% sigma7 = abs(sigma7);

%class 8
sigma8 = 0;
for i = 1996:2239
    sigma8 = sigma8 + (trainDataSet9(i,:)'-mu8)*(trainDataSet9(i,:)'-mu8)';
end
sigma8 = sigma8/244;
% sigma8 = abs(sigma8);

%class 9
sigma9 = 0;
for i = 2240:2499
    sigma9 = sigma9 + (trainDataSet9(i,:)'-mu9)*(trainDataSet9(i,:)'-mu9)';
end
sigma9 = sigma9/260;
% sigma9 = abs(sigma9);
```

```
% testSet classification
test_class = [];

for i = 1:2499
    x = testDataSet9(i,:);
    y0 = 1/((((2*pi)^9)*det(sigma0))^1/2)*exp(-0.5*(x-mu0)*inv(sigma0)*(x-
mu0)');
    y1 = 1/((((2*pi)^9)*det(sigma1))^1/2)*exp(-0.5*(x-mu1)*inv(sigma1)*(x-
mu1)');
    y2 = 1/((((2*pi)^9)*det(sigma2))^1/2)*exp(-0.5*(x-mu2)*inv(sigma2)*(x-
mu2)');
    y3 = 1/((((2*pi)^9)*det(sigma3))^1/2)*exp(-0.5*(x-mu3)*inv(sigma3)*(x-
mu3)');
    y4 = 1/((((2*pi)^9)*det(sigma4))^1/2)*exp(-0.5*(x-mu4)*inv(sigma4)*(x-
mu4)');
    y5 = 1/((((2*pi)^9)*det(sigma5))^1/2)*exp(-0.5*(x-mu5)*inv(sigma5)*(x-
mu5)');
    y6 = 1/((((2*pi)^9)*det(sigma6))^1/2)*exp(-0.5*(x-mu6)*inv(sigma6)*(x-
mu6)');
    y7 = 1/((((2*pi)^9)*det(sigma7))^1/2)*exp(-0.5*(x-mu7)*inv(sigma7)*(x-
mu7)');
    y8 = 1/((((2*pi)^9)*det(sigma8))^1/2)*exp(-0.5*(x-mu8)*inv(sigma8)*(x-
mu8)');
    y9 = 1/((((2*pi)^9)*det(sigma9))^1/2)*exp(-0.5*(x-mu9)*inv(sigma9)*(x-
mu9)');

    y_test_vec = [y0 y1 y2 y3 y4 y5 y6 y7 y8 y9];
    [maxx1,test_ind] = max(y_test_vec);
    test_class = [test_class test_ind-1];
end

test_correct = sum(test_class == testLabel')/2499*100

% trainSet classification
train_class = [];

for j = 1:2499
    x = trainDataSet9(j,:);
    y0 = 1/((((2*pi)^9)*det(sigma0))^1/2)*exp(-0.5*(x-mu0)*pinv(sigma0)*(x-
mu0)');
    y1 = 1/((((2*pi)^9)*det(sigma1))^1/2)*exp(-0.5*(x-mu1)*pinv(sigma1)*(x-
mu1)');
    y2 = 1/((((2*pi)^9)*det(sigma2))^1/2)*exp(-0.5*(x-mu2)*pinv(sigma2)*(x-
mu2)');
    y3 = 1/((((2*pi)^9)*det(sigma3))^1/2)*exp(-0.5*(x-mu3)*pinv(sigma3)*(x-
mu3)');
    y4 = 1/((((2*pi)^9)*det(sigma4))^1/2)*exp(-0.5*(x-mu4)*pinv(sigma4)*(x-
mu4)');
    y5 = 1/((((2*pi)^9)*det(sigma5))^1/2)*exp(-0.5*(x-mu5)*pinv(sigma5)*(x-
mu5)');
    y6 = 1/((((2*pi)^9)*det(sigma6))^1/2)*exp(-0.5*(x-mu6)*pinv(sigma6)*(x-
mu6)');
    y7 = 1/((((2*pi)^9)*det(sigma7))^1/2)*exp(-0.5*(x-mu7)*pinv(sigma7)*(x-
mu7)');
    y8 = 1/((((2*pi)^9)*det(sigma8))^1/2)*exp(-0.5*(x-mu8)*pinv(sigma8)*(x-
mu8)');
    y9 = 1/((((2*pi)^9)*det(sigma9))^1/2)*exp(-0.5*(x-mu9)*pinv(sigma9)*(x-
mu9)');
```

```
    y_train_vec = [y0 y1 y2 y3 y4 y5 y6 y7 y8 y9];
    [maxx2,train_ind] = max(y_train_vec);
    train_class = [train_class train_ind-1];
end

train_correct = sum(train_class == trainLabel')/2499*100

test_correct_vec = [19.7279 20.5282 24.1697 23.9696 18.3673 23.8095 25.4886
30.5722 34.7739];
train_correct_vec = [19.0876 20.3281 22.8091 23.7695 17.0068 22.8091
28.6883 34.4138 36.8547];
dimension_vec = [1 2 3 4 5 6 7 8 9];

figure;
plot(dimension_vec,test_correct_vec);
grid on
xlabel("Feature Dimension Size");
ylabel("Accuracy");
hold on
plot(dimension_vec,train_correct_vec);
grid on
title("LDA - Success Rate of Gaussian Classifier");
xlabel("Feature Dimension Size");
ylabel("Accuracy");
legend("Test Dataset","Train Dataset");

figure;
plot(dimension_vec,100-test_correct_vec);
grid on
xlabel("Feature Dimension Size");
ylabel("Accuracy");
hold on
plot(dimension_vec,100-train_correct_vec);
grid on
title("LDA - Error Rate of Gaussian Classifier");
xlabel("Feature Dimension Size");
ylabel("Accuracy");
legend("Test Dataset","Train Dataset");
```

sammon.m

```
function [y, E] = sammon(x, n, opts)
%SAMMON Performs Sammon's MDS mapping on dataset X
%
%    Y = SAMMON(X) applies Sammon's nonlinear mapping procedure on
%    multivariate data X, where each row represents a pattern and each
column
%    represents a feature.  On completion, Y contains the corresponding
%    co-ordinates of each point on the map.  By default, a two-dimensional
%    map is created.  Note if X contains any duplicated rows, SAMMON will
%    fail (ungracefully).
%
%    [Y,E] = SAMMON(X) also returns the value of the cost function in E
(i.e.
%    the stress of the mapping).
%
%    An N-dimensional output map is generated by Y = SAMMON(X,N) .
%
%    A set of optimisation options can also be specified using a third
```

```
%    argument, Y = SAMMON(X,N,OPTS) , where OPTS is a structure with
fields:
%
%       MaxIter        - maximum number of iterations
%       TolFun         - relative tolerance on objective function
%       MaxHalves      - maximum number of step halvgings
%       Input          - {'raw','distance'} if set to 'distance', X is
%                        interpreted as a matrix of pairwise distances.
%       Display        - {'off', 'on', 'iter'}
%       Initialisation - {'pca', 'random'}
%
%    The default options structure can be retrieved by calling SAMMON with
%    no parameters.
%
%    References :
%
%       [1] Sammon, John W. Jr., "A Nonlinear Mapping for Data Structure
%           Analysis", IEEE Transactions on Computers, vol. C-18, no. 5,
%           pp 401-409, May 1969.
%
%    See also : SAMMON_TEST

%
% File        : sammon.m
%
% Date        : Monday 12th November 2007.
%
% Author      : Gavin C. Cawley and Nicola L. C. Talbot
%
% Description : Simple vectorised MATLAB implementation of Sammon's non-
linear
%               mapping algorithm [1].
%
% References  : [1] Sammon, John W. Jr., "A Nonlinear Mapping for Data
%                   Structure Analysis", IEEE Transactions on Computers,
%                   vol. C-18, no. 5, pp 401-409, May 1969.
%
% History     : 10/08/2004 - v1.00
%               11/08/2004 - v1.10 Hessian made positive semidefinite
%               13/08/2004 - v1.11 minor optimisation
%               12/11/2007 - v1.20 initialisation using the first n
principal
%                                  components.
%
% Thanks      : Dr Nick Hamilton (nick@maths.uq.edu.au) for supplying the
%               code for implementing initialisation using the first n
%               principal components (introduced in v1.20).
%
% To do       : The current version does not take advantage of the symmetry
%               of the distance matrix in order to allow for easy
%               vectorisation.  This may not be a good choice for very
large
%               datasets, so perhaps one day I'll get around to doing a MEX
%               version using the BLAS library etc. for very large
datasets.
%
% Copyright   : (c) Dr Gavin C. Cawley, November 2007.
%
%    This program is free software; you can redistribute it and/or modify
%    it under the terms of the GNU General Public License as published by
```

```
%    the Free Software Foundation; either version 2 of the License, or
%    (at your option) any later version.
%
%    This program is distributed in the hope that it will be useful,
%    but WITHOUT ANY WARRANTY; without even the implied warranty of
%    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
%    GNU General Public License for more details.
%
%    You should have received a copy of the GNU General Public License
%    along with this program; if not, write to the Free Software
%    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
USA

% This file is part of the Matlab Toolbox for Dimensionality Reduction.
% The toolbox can be obtained from http://homepage.tudelft.nl/19j49
% You are free to use, change, or redistribute this code in any way you
% want for non-commercial purposes. However, it is appreciated if you
% maintain the name of the original author.
%a
% (C) Laurens van der Maaten, Delft University of Technology


% use the default options structure
if nargin < 3
    opts.Display        = 'iter';
    opts.Input          = 'raw';
    opts.MaxHalves      = 20;
    opts.MaxIter        = 2000;
    opts.TolFun         = 1e-9;
    opts.Initialisation = 'random';

end

% the user has requested the default options structure
if nargin == 0
    y = opts;
    return;
end

% Create a two-dimensional map unless dimension is specified
if nargin < 2
    n = 2;
end

% Set level of verbosity
if strcmp(opts.Display, 'iter')
    display = 2;
elseif strcmp(opts.Display, 'on')
    display = 1;
else
    display = 0;
end

% Create distance matrix unless given by parameters
if strcmp(opts.Input, 'distance')
    D = x;
else
    D = euclid(x, x);
end
```

```
% Remaining initialisation
N     = size(x, 1);
scale = 0.5 / sum(D(:));
D     = D + eye(N);
Dinv  = 1 ./ D;
if strcmp(opts.Initialisation, 'pca')
    [UU,DD] = svd(x);
    y       = UU(:,1:n)*DD(1:n,1:n);
else
    y = randn(N, n);
end
one   = ones(N,n);
d     = euclid(y,y) + eye(N);
dinv  = 1./d;
delta = D - d;
E     = sum(sum((delta.^2).*Dinv));

% Get on with it
for i=1:opts.MaxIter

    % Compute gradient, Hessian and search direction (note it is actually
    % 1/4 of the gradient and Hessian, but the step size is just the ratio
    % of the gradient and the diagonal of the Hessian so it doesn't
    % matter).
    delta    = dinv - Dinv;
    deltaone = delta * one;
    g        = delta * y - y .* deltaone;
    dinv3    = dinv .^ 3;
    y2       = y .^ 2;
    H        = dinv3 * y2 - deltaone - 2 * y .* (dinv3 * y) + y2 .* (dinv3
* one);
    s        = -g(:) ./ abs(H(:));
    y_old    = y;

    % Use step-halving procedure to ensure progress is made
    for j=1:opts.MaxHalves
        y(:) = y_old(:) + s;
        d     = euclid(y, y) + eye(N);
        dinv  = 1 ./ d;
        delta = D - d;
        E_new = sum(sum((delta .^ 2) .* Dinv));
        if E_new < E
            break;
        else
            s = 0.5*s;
        end
    end

    % Bomb out if too many halving steps are required
    if j == opts.MaxHalves
        warning('MaxHalves exceeded. Sammon mapping may not converge...');
    end

    % Evaluate termination criterion
    if abs((E - E_new) / E) < opts.TolFun
        if display
            fprintf(1, 'Optimisation terminated - TolFun exceeded.\n');
        end
```

```
        break;
    end

    % Report progress
    E = E_new;
    if display > 1
        fprintf(1, 'epoch = %d : E = %12.10f\n', i, E * scale);
    end
end

% Fiddle stress to match the original Sammon paper
E = E * scale;
end

function d = euclid(x, y)
d = sqrt(sum(x.^2,2)*ones(1,size(y,1))+ones(size(x,1),1)*sum(y.^2,2)'-
2*(x*y'));
end
```

sammonmapping.m

```
clear all; clc;
close all;

load digits.mat;

labelCount = [];
for i = 0:9
    labelc = labels == i;
    labelCount = [labelCount sum(labelc)];
end

% sorting data and labels in increasing order
[labels,isort] = sort(labels);
digits = digits(isort,:);

rng default % for reproducibility
Y = sammon(digits)
% load sammonsonunda.mat;
gscatter(Y(:,1),Y(:,2),labels);
title("2D Scatter Matrix with Class Informations - Sammon's Mapping");
xlabel("First Dimension");
ylabel("Second Dimension");
```

tsnetester.m

```
clear all; clc;
close all;

load digits.mat;

labelCount = [];
for i = 0:9
    labelc = labels == i;
    labelCount = [labelCount sum(labelc)];
end
```

```
% sorting data and labels in increasing order
[labels,isort] = sort(labels);
digits = digits(isort,:);

% rng default % for reproducibility
rng default % for reproducibility
Y = tsne(digits,'Algorithm','barneshut','NumPCAComponents',50);
gscatter(Y(:,1),Y(:,2),labels);
title("2D Scatter Matrix with Class Informations - tSNE");
xlabel("First Dimension");
ylabel("Second Dimension");
```