Lab 7: Face recognition using LDAs

In this lab, we will employ LDA and SVM to identify the faces from different people.

We will use the ORL database, available on AT&T's web site, as in previous labs. This database contains photographs showing the faces of 40 people (s1, s2, . . . , s40). Each one of them was photographed 10 times. These photos are stored as images in levels of grey with 112x92 pixels. The data has been split in two parts: train and test. For each person, we use the first 9 photographs for training and the last photograph for test.

1. Load the training and the testing data (separately) and change each (d1 = 112) x (d2 = 92) photograph into a vector;

```
% You can create a function on the form:
[trainingdata,testingdata]=loadImagesLab();
```

Classification using LDA

Apply LDA to the **training** set. Follow these steps:

2. Estimate the between class covariance and within class covariance. Then, use them to estimate the LDA matrix *V* that solves the following *generalized* Eigen problem

Sb.*v* = l.*Sw*.*v* (use the eigs Matlab function [V, D]=eigs(*SB* , *Sw* , L-1) to compute the eigenvectors), where *SB* contains the between class scatter and *Sw*, the within class scatter of the training images feature vectors (you can use PCA weights as features for training and testing). Remember that if you have *L* different classes, LDA will provide *L-1* non-zero eigenvalues.

```
%You can create several functions on the form:
PCAdimension = 64;
[TrainingPCA,TestingPCA]=PCAlab(trainingdata,testingdata,PCAdimension);
```

```
LDAdimension = 39;
[V,D,TrainingLDA,TestingLDA]=LDAlab(TrainingPCA,TestingPCA,LDAdimension);
```

3. Check the identity corresponding to each photograph in the **testing** set by determining its LDA projection (with dimension 10, 20, 30 and 39) employing the LDA matrix obtaining with the training data and then comparing the distances of this projection with respect to all projections in the training data. For each test category find the closest category in the test. Report your accuracy

```
%You can create several functions on the form:
% [LDAresults]=distancesLab(TrainingLDA,TestingLDA);
% Also, you can generate figures representing the accuracy as a function of the PCAdime
```

```
[LDAresults ] = distancesLab(TrainingLDA, TestingLDA);
```

```
LDA Test Set Accuracy: 0.775000
```

```
% so lets make a grid search
pcadims = [64, 96, 128, 256];
ldadims = [10, 20, 30, 39];
grid = zeros(4, 4);

gridsvm = zeros(4, 4);
for i = 1:length(pcadims);
    [trainPCA, testPCA] = PCAlab(trainingdata, testingdata, pcadims(i));
    for j = 1:length(ldadims)
        if ldadims(j) > pcadims(i)
            continue
        end
        [V,D,trainLDA, testLDA] = LDAlab(trainPCA, testPCA, ldadims(j));
        [LDAresults ] = distancesLab(trainLDA, testLDA);
        accuracy = mean(LDAresults == [1:40]');
        grid(i, j) = accuracy;
        fprintf("Grid(PCA %d,LDA %d) -- Distance Accuracy is %f\n", ...
            pcadims(i), ldadims(j), accuracy);
    end
end
```

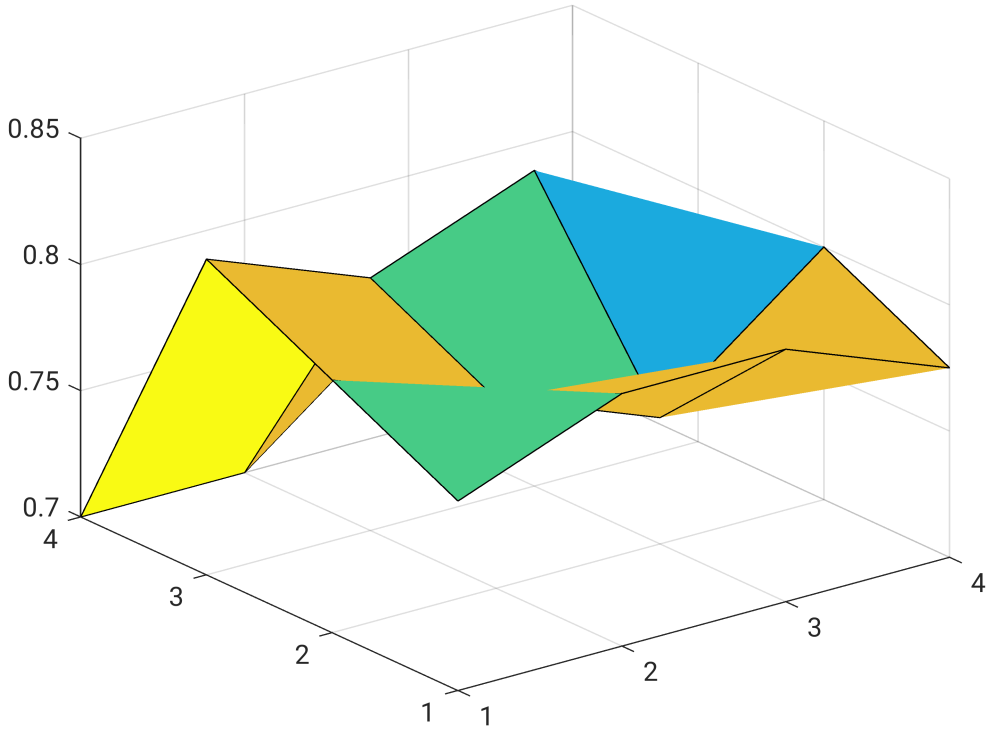
```
LDA Test Set Accuracy: 0.775000
Grid(PCA 64,LDA 10) -- Distance Accuracy is 0.775000
LDA Test Set Accuracy: 0.800000
Grid(PCA 64,LDA 20) -- Distance Accuracy is 0.800000
LDA Test Set Accuracy: 0.800000
Grid(PCA 64,LDA 30) -- Distance Accuracy is 0.800000
LDA Test Set Accuracy: 0.775000
Grid(PCA 64,LDA 39) -- Distance Accuracy is 0.775000
LDA Test Set Accuracy: 0.800000
Grid(PCA 96,LDA 10) -- Distance Accuracy is 0.800000
LDA Test Set Accuracy: 0.775000
Grid(PCA 96,LDA 20) -- Distance Accuracy is 0.775000
LDA Test Set Accuracy: 0.750000
Grid(PCA 96,LDA 30) -- Distance Accuracy is 0.750000
LDA Test Set Accuracy: 0.800000
Grid(PCA 96,LDA 39) -- Distance Accuracy is 0.800000
LDA Test Set Accuracy: 0.825000
Grid(PCA 128,LDA 10) -- Distance Accuracy is 0.825000
LDA Test Set Accuracy: 0.800000
Grid(PCA 128,LDA 20) -- Distance Accuracy is 0.800000
LDA Test Set Accuracy: 0.825000
Grid(PCA 128,LDA 30) -- Distance Accuracy is 0.825000
LDA Test Set Accuracy: 0.775000
Grid(PCA 128,LDA 39) -- Distance Accuracy is 0.775000
LDA Test Set Accuracy: 0.700000
Grid(PCA 256,LDA 10) -- Distance Accuracy is 0.700000
LDA Test Set Accuracy: 0.700000
Grid(PCA 256,LDA 20) -- Distance Accuracy is 0.700000
LDA Test Set Accuracy: 0.750000
Grid(PCA 256,LDA 30) -- Distance Accuracy is 0.750000
LDA Test Set Accuracy: 0.775000
Grid(PCA 256,LDA 39) -- Distance Accuracy is 0.775000
```

```
grid
```

```
grid = 4×4
    0.7750    0.8000    0.8000    0.7750
    0.8000    0.7750    0.7500    0.8000
    0.8250    0.8000    0.8250    0.7750
    0.7000    0.7000    0.7500    0.7750
```

```
[x, y] = meshgrid(1:4);
surf(x, y, grid);
```

```
Warning: MATLAB has disabled some advanced graphics rendering features by switching to software
OpenGL. For more information, click here.
```




```
fprintf("peak performance at (PCA 128, LDA 10) and (PCA 128, LDA 30). Accuracy 0.825.")
```

```
peak performance at (PCA 128, LDA 10) and (PCA 128, LDA 30). Accuracy 0.825.
```

```

```

```

```

Optional: Classification using LDA+SVM (+5 extra points)

4. Using the LDA projections of the training faces, train a SVM model to identify the different people. Therefore, resulting model will include 40 classes. Then, test the model using the LDA projections of the testing faces. Use the Matlab function or LIBSVM library to do the binary classification. Examples can be found in https://

www.mathworks.com/help/stats/support-vector-machine-classification.html for Matlab. Play with different C values and kernel functions and see how they influence the result. Report your best accuracy and settings include dimension, C value, kernel function you used.

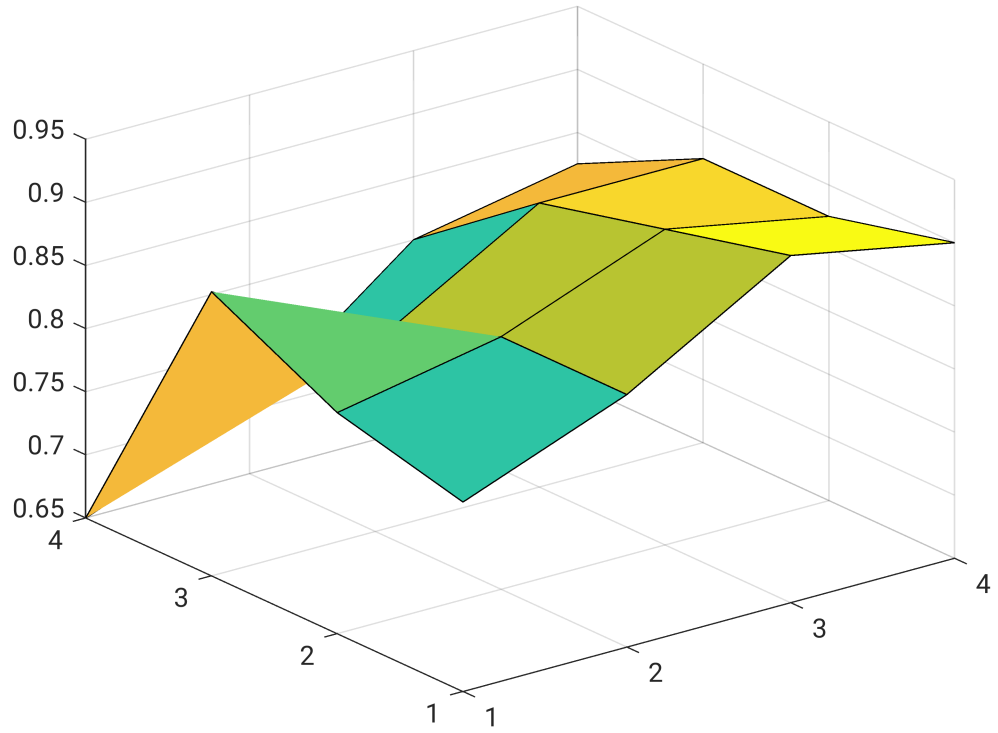
```
% [SVMresults]=SVMlab(TrainingLDA,TestingLDA);
% mean(SVMresults == [1:40]')
% so lets make a grid search
pcadims = [64, 96, 128, 256];
ldadims = [10, 20, 30, 39];

gridsvm = zeros(4, 4);
for i = 1:length(pcadims);
    [trainPCA, testPCA] = PCAlab(trainingdata, testingdata, pcadims(i));
    for j = 1:length(ldadims)
        if ldadims(j) > pcadims(i)
            continue
        end
        [V,D,trainLDA, testLDA] = LDAlab(trainPCA, testPCA, ldadims(j));
        [SVMresults ] = SVMlab(trainLDA, testLDA);
        accuracy = mean(SVMresults == [1:40]');
        gridsvm(i, j) = accuracy;
        fprintf("Grid(PCA %d,LDA %d) -- SVM Accuracy is %f\n", ...
           pcadims(i), ldadims(j), accuracy);
    end
end
```

```
Grid(PCA 64,LDA 10) -- SVM Accuracy is 0.800000
Grid(PCA 64,LDA 20) -- SVM Accuracy is 0.850000
Grid(PCA 64,LDA 30) -- SVM Accuracy is 0.925000
Grid(PCA 64,LDA 39) -- SVM Accuracy is 0.900000
Grid(PCA 96,LDA 10) -- SVM Accuracy is 0.825000
Grid(PCA 96,LDA 20) -- SVM Accuracy is 0.850000
Grid(PCA 96,LDA 30) -- SVM Accuracy is 0.900000
Grid(PCA 96,LDA 39) -- SVM Accuracy is 0.875000
Grid(PCA 128,LDA 10) -- SVM Accuracy is 0.875000
Grid(PCA 128,LDA 20) -- SVM Accuracy is 0.800000
Grid(PCA 128,LDA 30) -- SVM Accuracy is 0.875000
Grid(PCA 128,LDA 39) -- SVM Accuracy is 0.875000
Grid(PCA 256,LDA 10) -- SVM Accuracy is 0.650000
Grid(PCA 256,LDA 20) -- SVM Accuracy is 0.700000
Grid(PCA 256,LDA 30) -- SVM Accuracy is 0.800000
Grid(PCA 256,LDA 39) -- SVM Accuracy is 0.825000
```

```
gridsvm
```

```
gridsvm = 4×4
    0.8000    0.8500    0.9250    0.9000
    0.8250    0.8500    0.9000    0.8750
    0.8750    0.8000    0.8750    0.8750
    0.6500    0.7000    0.8000    0.8250
```

```
surf(x, y, gridsvm);
```




```
fprintf('performance peaks at (PCA 64, LDA 30). Accuracy 0.925');
```

```
performance peaks at (PCA 64, LDA 30). Accuracy 0.925
```

```
fprintf("just default parameters of the fitcecoc function.")
```

```
just default parameters of the fitcecoc function.
```

```
% You can generate figures representing the accuracy as a function of the PCAdimension
```

Provide a report in *pdf* format explaining your results (you can provide a pdf of this script including graphics and results).

NOTES:

- The suggested functions are just a guide. You can create them or not. Also, you can add or remove input/output parameters as needed.
- Between class scatter:

$$S_B = \sum_{k=1}^{K} N_k (m_k - m)(m_k - m)^T$$

where *K* is the total number of classes, N_k is the number of observations on class *k*, $m_k$is the mean of vectors of class *k* and m is the mean of all the vectors (global mean).

- Within-class scatter:

$$S_w = \sum_{k=1}^{K} \sum_{i=1}^{N_k} (x_{ik} - m_k)(x_{ik} - m_k)^T$$

where x_{ik} is the i^{th} observation of class *k*.

- PCA projection is recommended to obtain the features to be used to train the LDA matrix in order to avoid singularity problems. If PCA is not used over the original data, the number of features per observation (10304) will be much larger than the number of observations (360) and the covariance matrices will not have full rank and will not be invertible.