COM2004/3004 Lab Week 11 – Face Recognition

Objective: In this lab you will use PCA and a Nearest Neighbour classifier to tackle a challenging classification task – face recognition.

Background

In an earlier lab class you used PCA and a nearest neighbour classifier to tackle a very simple character recognition task. In this lab you will be using the same techniques but applying them to a more challenging task – face recognition. You are provided with a small data set containing 10 images each of 40 different people, i.e. 400 images in total. You will be splitting this into training data and test data. The classification system will be built using only the training data and then tested on the unseen test data. A jackknife training/testing schedule will be used to reliably test the performance of the system.

1. Download the data and tools

Make a folder for the lab class and download the following files from MOLE week 11.

`faces.tif` - the face image data. 400 images stored as one big image.
`loadfaces.m` – code for loading in the face data.
`learnPCA.m` – code for performing PCA analysis.
`reducePCA.m` – code to perform the PCA dimensionality reduction.
`computePCAaxes.m` – a helper function for `learnPCA.m`.
`classify.m` – the nearest neighbor classifier that we've used before.

2. Loading the face data

The face data is stored as one big image, `faces.tif`. If you try opening this in Windows you should be able to view it. I have provided a program that segments this image and extracts each of the 400 faces contained in it as a separate feature vector. The individual face images have dimension 46 rows by 56 columns giving the raw feature vectors a dimensionality of 2,576.

Type,

```
data = loadfaces();
```

This will construct a 400 by 2,576 matrix. To view a single face as an image use,

```
imagesc(reshape(data(1,:), 56, 46));
colormap(gray);
```

You should see that rows 1 to 10 are images of person 1; rows 11 to 20 are person 2; rows 21 to 30 are person 3 and so on up to person 40.

3. Dividing the data into training and testing

(Tip: you will be reusing the bits of MATLAB in sections 3 and 4 when you come to section 5, so it will save you a lot of retyping if you save them in a script file as you go.)

We are going to use 9 examples of each person to train the system and 1 example of each person to test the system. We will start by making training and testing labels. We will use an integer coding, i.e. 1 for person 1; 2 for person 2; 3 for person 3 and so on.

Making the test labels is easy as there is going to be just one example of each person so,

```
test_labels = 1:40;
```

The training labels need to be a vector containing nine 1s followed by nine 2s and then nine 3s and so on. We could type it directly, but the following hacky bit of MATLAB will do the job,

```
train_labels = reshape(repmat(1:40,9,1),1,360);
```

Now we need to split up the data itself. We will choose the test data to be the *K*th example of each person (where *K* is from 1 to 10) and the training data will be all the rest. So for example if *K*=1 the test data would be rows 1, 11, 21, ...391 of `data` and training data will be the remaining 360 rows.

How can we do this neatly? The following achieves the desired result by using a modulus operation,

```
K=1;
train_data = data((mod(1:400,10)+1)~=K,:);
test_data = data((mod(1:400,10)+1)==K,:);
```

(Make sure that you understand why the above instructions work).

4. Performing the dimensionality reduction

We are now going to perform the PCA analysis and use the analysis to reduce the dimensionality of the data. I've split the PCA dimensionality reduction into two stages: `learnPCA.m` will analyse the training data and find the first *N* principle component axes. `reducePCA.m` will then be used to reduce the dimensionality of the training data and test data by projecting onto these *N* axes. It is logical to split it this way because the analysis step is only applied to the training data, whereas the dimensionality reduction will be applied to all the data.

First perform the analysis step,

```
nAxes = 10;
```

```
    [pca_axes, mean_vector] = learnPCA(train_data, nAxes);
```

Note the program also returns the mean_vector – we will need this when performing the projection.

To display one of the PCA axes as an image (i.e. an eigenface) you can do,

```
    PCA_N = 1;
    imagesc(reshape(pca_axes(PCA_N, :), 56, 46));
```

look at the first two or three eigenfaces and compare them to the pictures in the lecture notes.

The next step is to actually project all the data on to the *N* PCA axes, i.e. the dimensionality reduction. You can use the program `reducePCA` to do this,

```
reduced_train = reducePCA(train_data, pca_axes, mean_vector);
reduced_test = reducePCA(test_data, pca_axes, mean_vector);
```

It is worth looking at the code for `learnPCA` and `reducePCA` to make sure that it makes sense to you.

5. Performing the classification

We can now perform the classification in the PCA space using our nearest neighbor classifier code,

```
    [score, cm] =
    classify(reduced_train,train_labels,reduced_test,test_labels);
```

If all has gone well, the score should be well above 90%. Look also at the confusion matrix.

This is fine but we have only classified 40 images so the result is not very statistically significant and the confusion matrix is not very interesting because we haven't seen enough examples to get a measure of the common confusions.

The last step will be to repeat everything we have done so far 10 times where each time we test a different 10th of the data. Looking back at section 3, we can see that by setting `K=2` we can re-divide the 400 images so that every 2nd example of each person is now the test image and the remainder is the training data. If we placed a loop of K from 1 to 10 around all the instruction in section 3, 4 and 5 we could perform the testing 10 times each time using a different 10th of the data for testing. We would then simply average the 10 scores coming out of the classify command and sum up the 10 confusion matrices. This is called jackknife testing.

Write the loop and compute the overall score and confusion matrix.

Look at the confusion matrix. There are some interesting repeated confusions. Is the classifier making mistakes that you can imagine that you might make yourself? Now that you have the code in a script file try using a smaller number of PCA coefficients, or a larger number. How does classification performance vary?

JPB 20/11/2011