NA_08: Face Recognition by Egenfaces

Numerical algorithms, University of Konstanz, ST 2008

by Vladimir Bondarenko, 10-June.

In todays Lab, we learn how to apply the *principle component analysis* for classification purpose. The objects we are going to classify are our own faces.

Contents

0. Face data

Images are 2D arrays. Classical PCA, however, works on vector data, as we have seen in the Height-Weight example in the PCA and SVD Labs. Therefore before the analysis, the 2D images must be reshaped into *looong* 1D vectors. This can be done simply concatenating the *n* columns of the *m-by-n* image matrix. In MATLAB, this vectorization operation for a matrix **A** is just `a = A(:)`.

There are three face databases available: att, yale and konstanz. The first two are freely available in the Internet, the last one was created from the images of the participants of this course. You can specify the database, with which you would like to work, in the variable 'db':

```
db = 'att.mat';
db = 'yale.mat';
db = 'konstanz.mat';
```

To load the face data into MATLAB workspace just run: `load(db)`. In the workspace, three variables will appear:

- `F` - the Face matrix, which rows contain the vectorized face images;
- `N` - the number of subjects (persons) in the data base;
- `m, n` - the size of a face image matrix (before vectorization).

```
db = 'konstanz';
load(db);
```

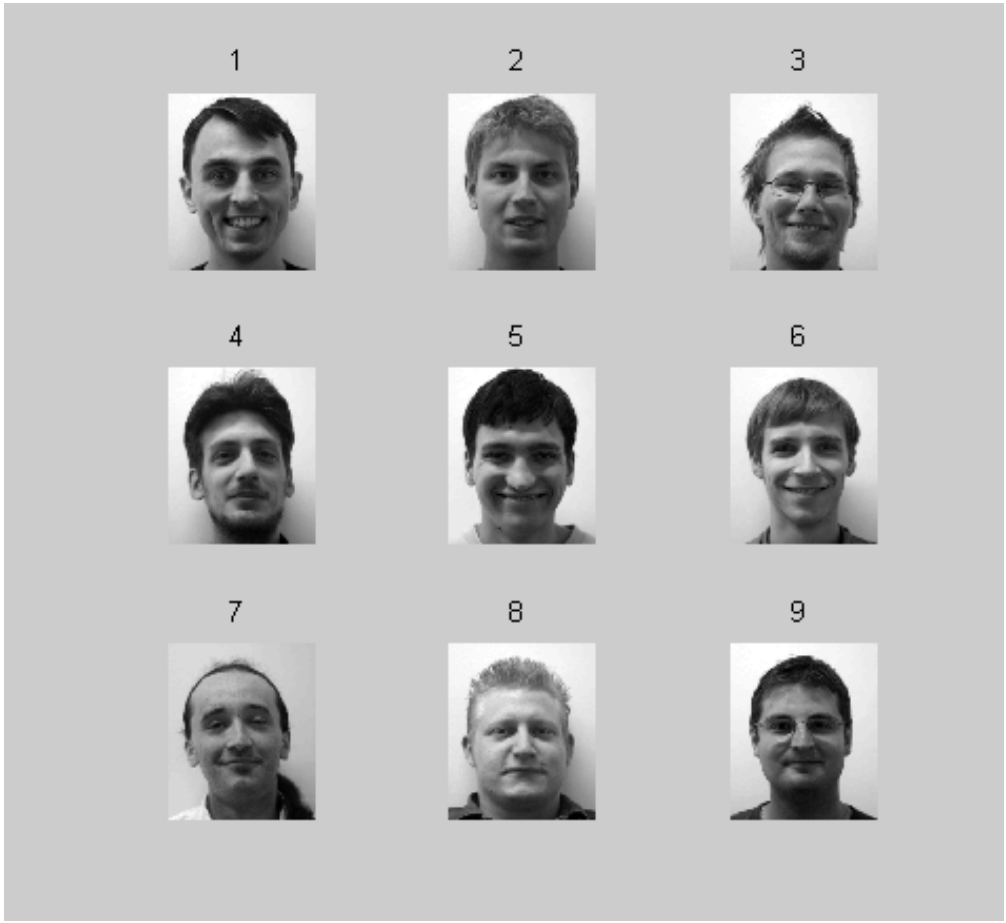
Explore the faces:

For each subject (person), several face images corresponding to different face expressions were collected. You can explore those by specifying the `exprNum` variable in this cell:

```
NN  = size(F,1);        % Overal number of face images.
fps = NN/N;             % Number of faces per subject.
```

```
% Display faces
exprNum = 2;            % The number of the particular face expression.

nrows = round(sqrt(N));
ncols = ceil(N/nrows);
figure(1); clf; set(gcf, 'Name', 'Faces');
for ii=1:N
    subplot(nrows,ncols,ii);
    tmp = (ii-1)*fps + exprNum;
    imagesc( reshape(F(tmp,:),m,n) );
    colormap gray; axis equal tight off;
    title(num2str(ii));
end
```



1. PCA

As it is often the case in applied math, we are interested in a better representation of our data, i.e., in a better basis (than the standard one). We assume that there is some kind of linear dependency among the face vectors and thus the *principle axes* obtained by PCA should provide a fairly good basis. In the following, we will check this hypothesis.

Task 1: SVD

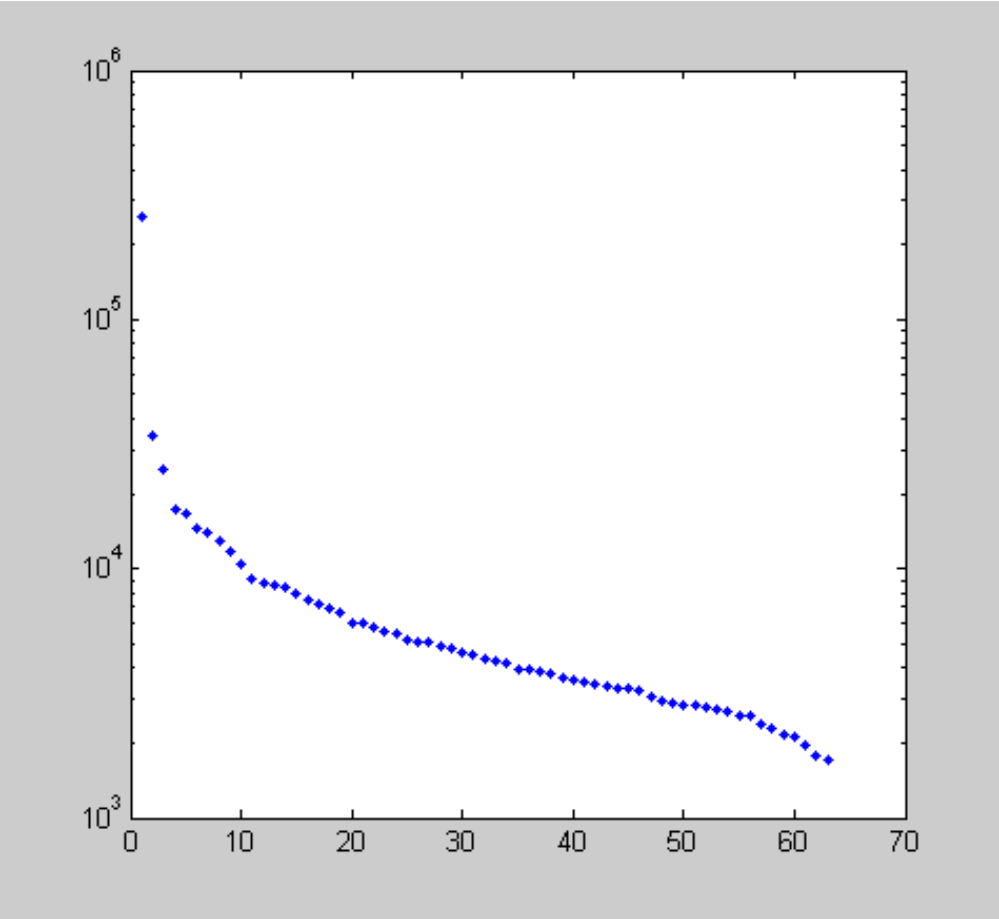
- Compute the SVD of the Face matrix, **F**;
- Plot the singular values on a logarithmic scale. Do they decrease sufficiently fast?;
- The columns of the SVD matrix **V** are the *principle axes* for the Face data. Compute the principle components (coordinates) of the face images w.r.t. this basis and store

them *row-wise* into a matrix **PC**.

Caution: the MATLAB `svd()` and `svd( ,0)` commands will compute square matrices **V** of dimension of several thousands. It is unnecessary since we are going to use only several first columns of **V**. Use the command `svd(,'econ')` instead to compute the first *N* columns of **V** only.

YOUR CODE HERE:

```
[U S V] = svd(F, 'econ');
PC = U*S;                                   % the principle components
figure(2); clf; semilogy(diag(S), '.');
```



Task 2: Explore the "eigenfaces"

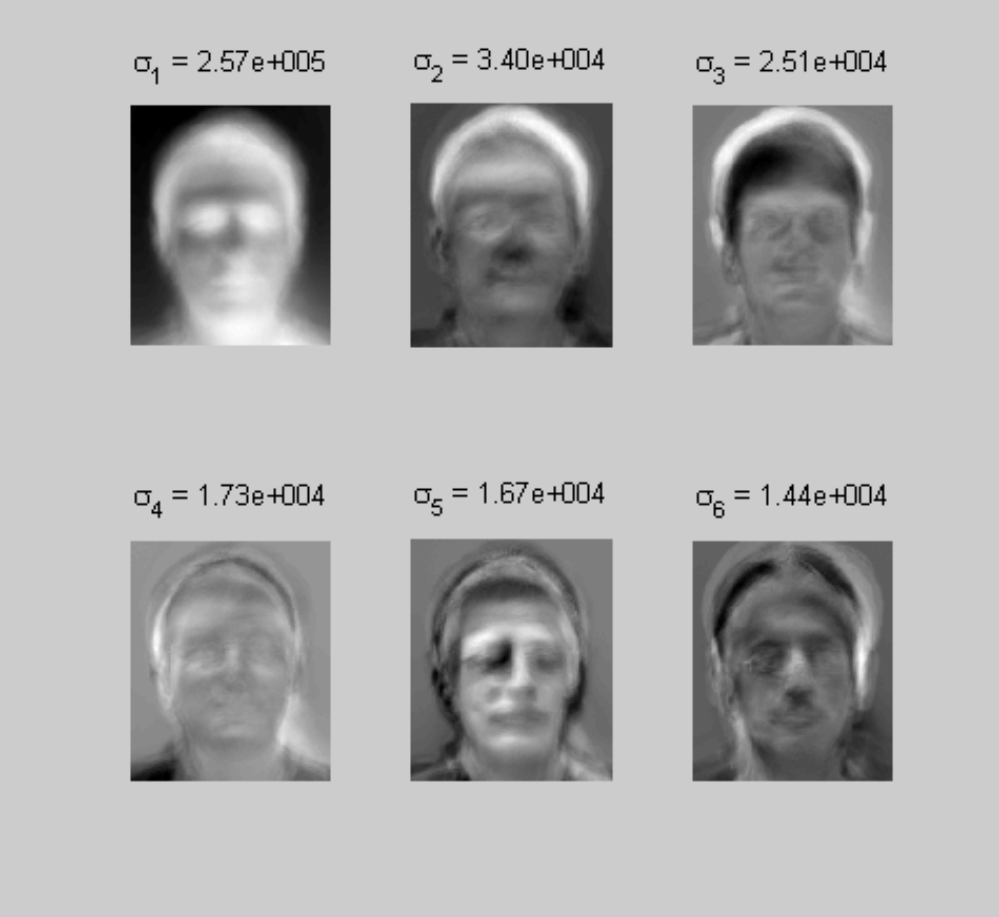
The new basis, the *principle axes*, have the same dimension that the original face images and thus can be reshaped and visualized as a matrix. The first 6 eigenface are displayed below. Can you recognise yourself in them?

- Make a plot of some eigenfaces yourself;
- Ask yourself, why the eigenfaces have this face-like appearance?

YOUR CODE HERE:

```
nrows = 2;
ncols = 3;
```

```
figure(3); clf; set(gcf, 'Name', 'EigenFaces');
for ii=1:nrows*ncols
    subplot(nrows,ncols,ii);
    imagesc( reshape(V(:,ii),m,n) );
    colormap gray; axis equal tight off
    title(['\sigma_' num2str(ii) ' = ' num2str(S(ii,ii), '%1.2e') ]);
end
```



2. Construct Eigenfaces subspace

The eigenfaces span the `N`-dimensional subspace where our Face data are. Howevere, according to our hypothesis of linear dependency, much less number of basis vectors must provide a reasonable representation of the Face data. Let us try to reduce the number of eigenfaces from `N` to nDim and see whether the nDim basis vectors indeed yield a satisfactory reconstrustruction.

```
nDim = 20;                  % dimensionality of the eigenfaces subspace
V1 = V(:,1:nDim);           % reduced basis of eigenfaces
```

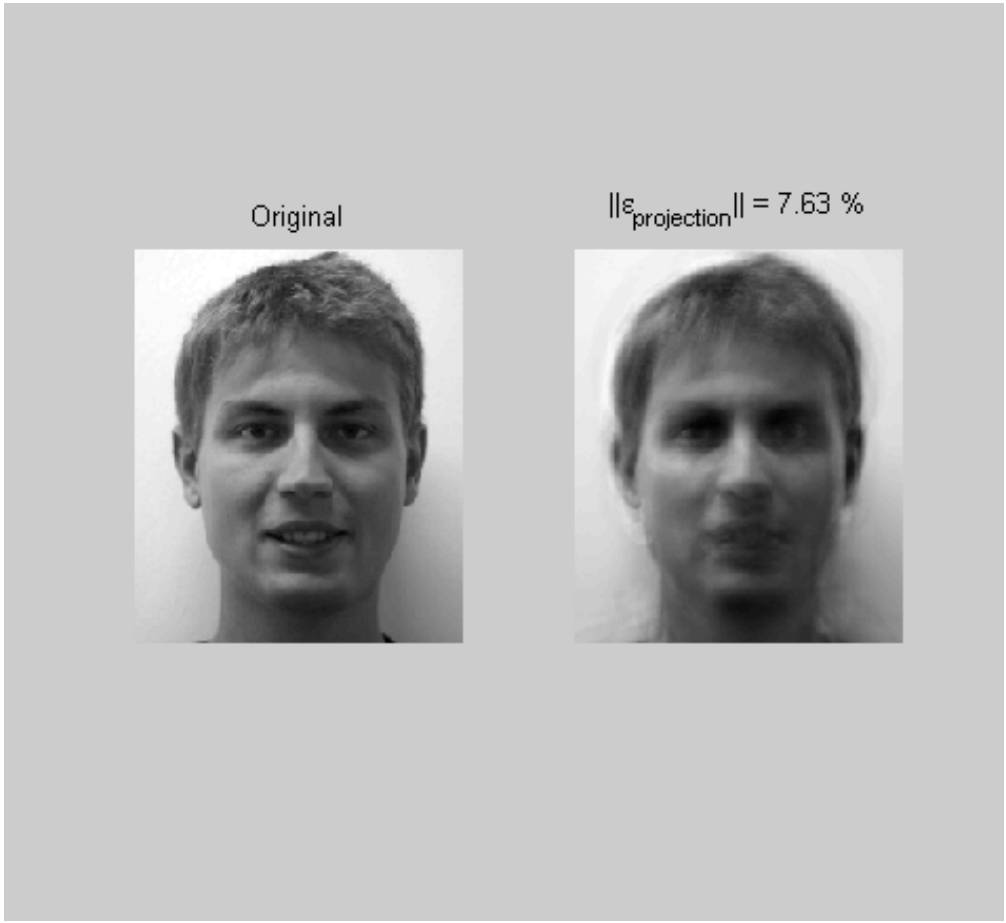
Task 3: Explore reconstructions

A random face `f` is selected from the Face matrix **F**.

- From the **PC** matrix, extract first `nDim` principle coordinates corresponding to `f` and compute its reconstruction, `f1`;
- Compute the relative rms error of the reconstruction, rmse;
- When done, run the cell and observe the quality of the reconstruction.

```
fn = randint(1,1,[1, NN]);            % Select face at random
f   = F(fn,:)';
% YOUR CODE HERE:
pc1 = PC(fn,1:nDim);                  % project it on the first nDim principle axes
f1 = V1*pc1';                         % reconstructed face
rmse = norm(f-f1)/norm(f);            % rms reprojection error

% Plot the original and reconstructed images
figure(4); clf;
subplot(1,2,1); imagesc(reshape(f,m,n)); title('Original');
axis equal tight off
subplot(1,2,2); imagesc(reshape(f1, m,n));
title(['||\epsilon_{projection}|| = ' num2str(rmse*100,'%2.2f') ' %']);
colormap gray; axis equal tight off
```



3. Construct the feature space

So far, each subject is associated with several nDim-vectors of principle components. Our goal is to define a single characteristic "feature vector" for each person. In the most straightforward way, such a vector can be constructed simply by averaging the corresponding principle component vectors. Those `N` `nDim`-feature vectors define the feature space that will be searched by the face recognition algorithms. Notice the enormous dimensionality reduction that occured when moving from the `m-times-n`-dimensional space of all images to the nDim-dimensional feature space.

You do not have to compute the feature vectors yourself. The code below does it for you. The resulting feature vectors are stored in the rows of the `N-by-nDim` dimensional matrix PC_mean.

The next plot shows the face images reconstructed from the feature vectors.

```
PC_mean = zeros(N, nDim);       % average feature vectors for each subject
for ii=1:N
    ix1 = fps*(ii-1)+1; ix2 = ix1+fps-1;
    PC_mean(ii,:) = mean(PC(ix1:ix2,1:nDim));
end
% Plot faces reconstructed from the feature vectors
nrows = round(sqrt(N));
ncols = ceil(N/nrows);
figure(1); set(gcf, 'Name', 'Face averages');
for ii=1:N
    subplot(nrows,ncols,ii);
    imagesc(reshape(V1*PC_mean(ii,:)',m,n));
    colormap gray; axis equal tight off
    title(num2str(ii));
end
```

4. Face recognition

After we have constructed the feature space, the algorithm for face recognition is straightforward.

For a (vectorized) face image *f*:

- Compute its feature vector, i.e., its coordinates in the (reduced) eigenface basis;
- Compute the distances between the feature vectors corresponding to *f* and to the subjects in our database;

- Find the nearest subject.

Task 4: Face recognition

- Implement the face recognition algorithm described above for a test face selected at random from our Face matrix;
- In a single figure, display the test and the recognized faces;
- Plot the inter-feature-vectors distances as a bar plot.

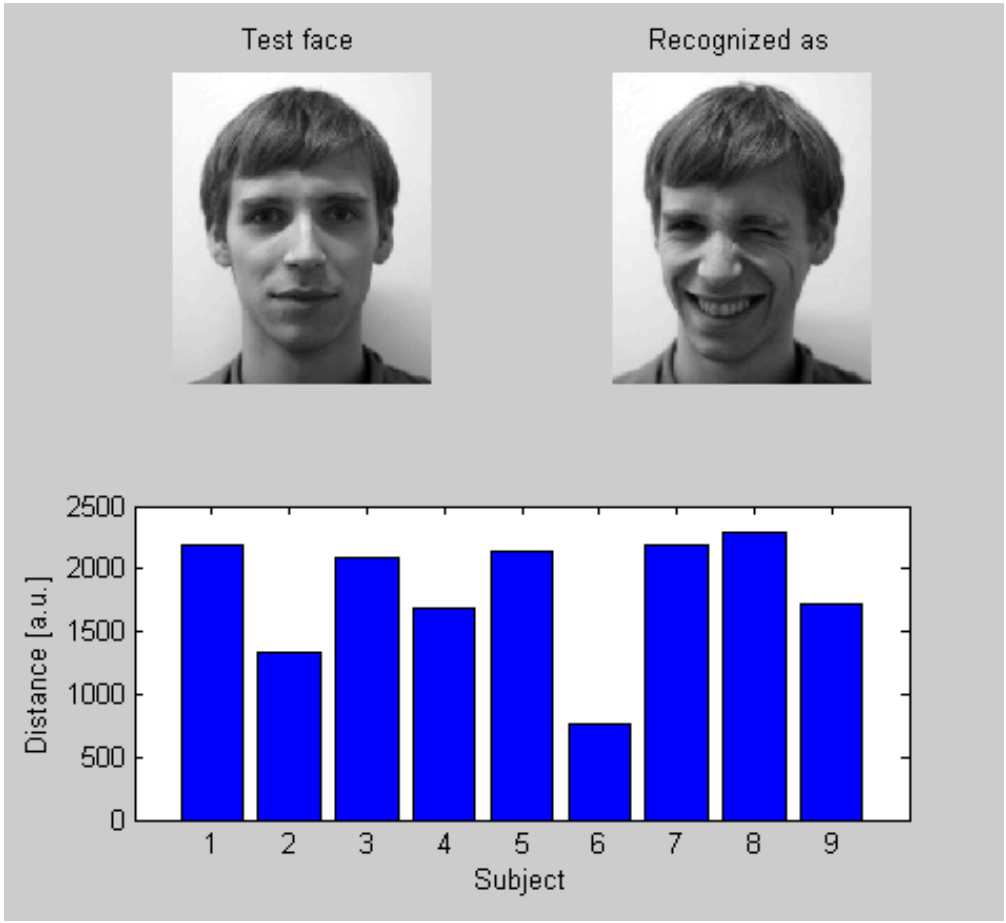
Your plot should look like the one below.

YOUR CODE HERE:

```
% Select a face at random and compute its feature vector:
fn = randint(1,1,[1, NN]);
f = F(fn,:);                          % test face
pc1 = V1'*f';                         % its principle component

% Compute distances in the feature space
Dist = PC_mean - ones(N,1)*pc1';
dist = sqrt(mean(Dist.^2, 2));
[min_dist, min_ix] = min(dist);

% Plot
figure(5); clf; set(gcf, 'Name', 'Face recognition')
subplot(2,2,1); imagesc( reshape(f,m,n) );
axis equal tight off; colormap gray;
title('Test face');
subplot(2,2,2); imagesc( reshape( F((min_ix-1)*fps+7,:), m,n) );
axis equal tight off; colormap gray;
title('Recognized as');
subplot(2,2,[3 4]); bar(dist, 'FaceColor', 'b');
xlabel('Subject'); ylabel('Distance [a.u.]');
```



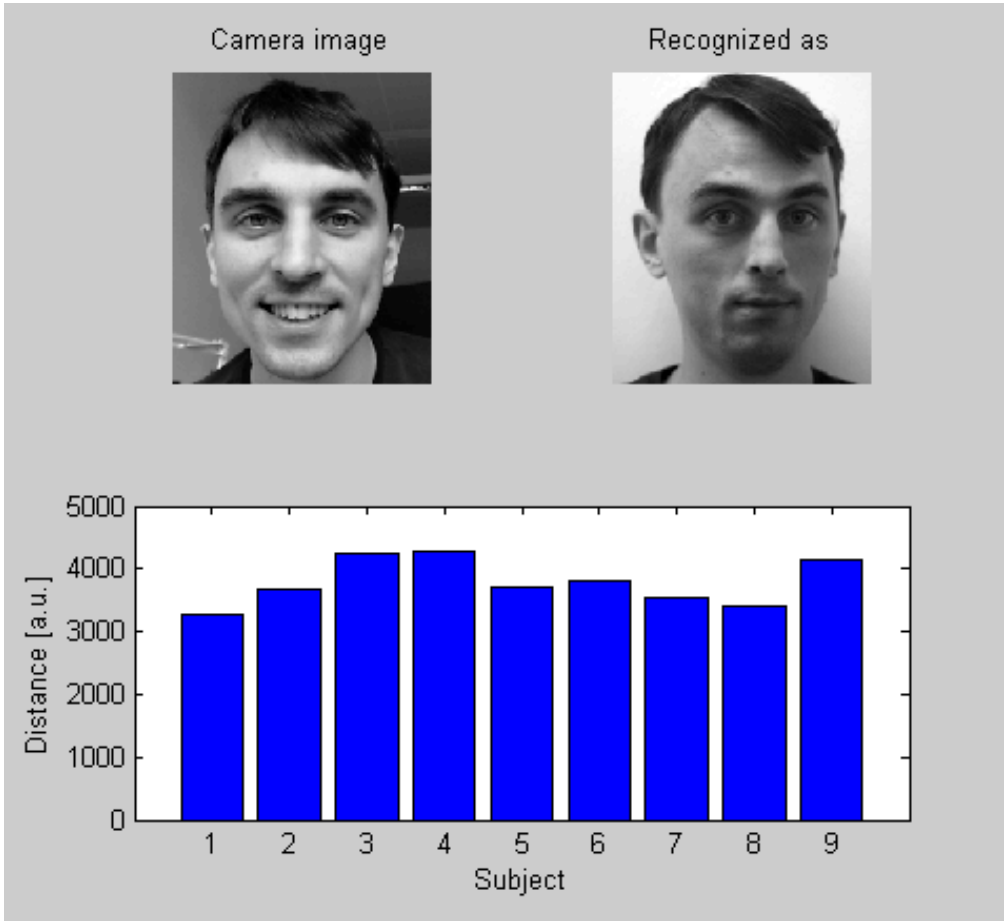
5. Real-time face recognition

In reality, the face recognition with this simple approach is not very reliable. Notice the distances in the plot below.

```
camF = double(t);
camf = camF(:);

pc1 = V1'*camf;                          % its principle component
% Compute distances in the feature space
Dist = PC_mean - ones(N,1)*pc1';
dist = sqrt(mean(Dist.^2, 2));
[min_dist, min_ix] = min(dist);

% Plot
figure(5); clf; set(gcf, 'Name', 'Face recognition')
subplot(2,2,1); imagesc( camF );
axis equal tight off; colormap gray;
title('Camera image');
subplot(2,2,2); imagesc( reshape( F((min_ix-1)*fps+5,:), m,n) );
axis equal tight off; colormap gray;
title('Recognized as');
subplot(2,2,[3 4]); bar(dist, 'facecolor', 'b');
xlabel('Subject'); ylabel('Distance [a.u.]');
```



Published with MATLAB® 7.6