Fast Prototyping Exercise 1

Exercises 5, 6, 7
CSC872
Pattern Analysis and Machine Intelligence

Fast Prototyping Exercise

- **Fast Prototyping**
  - Learn how to do a quick proof of concept by building a “prototype” (from papers you read, no public codes)
  - **Correctness** matters (no sloppy algorithm!)
  - **Speed** matters (no beautification!)
  - No perfect SE necessary
  - No copying of codes online.
  - **When Done: Parameterization/Visualization/Experimentation**
    - Find out what are **free parameters** in your algorithm whose value must be hand-picked by you
    - Learn how to view internal variable’s current values
    - Learn how to visualize your prototype’s results in plots/images etc
    - Tweak the parameter values and study your prototype’s behavior [quantitatively](#) to understand the how algorithm works

- **Group Work**
  - You are encouraged to freely exchange ideas and codes
  - Contributions to others are as valuable as making your own work

https://bidal.sfsu.edu/~kazokada/csc872/FaceRecognition_Data.zip
Fast Prototyping Exercise

- Please upload your matlab codes thru iLearn forum for my grading and your playing!
- Every week at the end of your excise, please submit your codes/scripts to the specified iLearn forum by midnight of the day. Your grade on FP exercise will be partly based on these submitted codes and what I observe during the in-class exercises.
- If received helps from others, please credit the person who helped you.

Platforms

- MATLAB
  - MathWorks: http://www.mathworks.com/

- MATLAB clones
  - Octave: http://www.gnu.org/software/octave/
  - SciLab: http://www.scilab.org/
Public Libraries

- **OpenCV (Computer Vision)**

- **ITK (Medical Imaging)**
  - [http://www.itk.org/](http://www.itk.org/)

- **WEKA (Machine Learning)**

Face Recognition by Eigenface

- Let’s create a face recognition system using one of the most basic algorithm called “**Eigenface**”.
  - You have not studied this in the lecture yet but
  - You read a paper on this (Turk & Pentland)

- You will need to implement
  - Image I/O + visualization
  - PCA for learning
  - Recognition by nearest neighbor classification
### Paper 1

- M. Turk, A. Pentland,

  - [http://portal.acm.org/citation.cfm?id=1326887.1326894&coll=&dl=](http://portal.acm.org/citation.cfm?id=1326887.1326894&coll=&dl=)

### Data

- I provide a set of facial images
- [https://bidal.sfsu.edu/~kazokada/csc872/FaceRecognition_Data.zip](https://bidal.sfsu.edu/~kazokada/csc872/FaceRecognition_Data.zip)
- Images are organized in 3 folders
- FA: 12 32x32 8bit facial images (for Known faces DB)
- FB: 23 facial images (for Test Set)
- ALL = FA+FB (for Training)
Principal Component Analysis

- **Conceptual Steps**
  1) Collect $M$ Training Images (must be aligned, $N \times N$ by $N$ matrix)
  2) Vectorize the Images: $X = \{x_1, \ldots, x_M\}$ Each of $M$ images is a column vector with $N$ coefficients where $N = N \times N$
  3) Compute mean image: $\mu = mean(X)$; a vector of $N$ coeffs
  4) Construct Covariance Matrix: $C = (X - \mu)(X - \mu)^T$ $N$ by $N$ mat
  5) Solve Eigenvalue Problem: $Cv_i = \lambda_i v_i$
  6) Sort resulting eigen vectors in decreasing order of corresponding eigen values.
  7) Select the top $K$ Eigenvectors $W = \{v_1, \ldots, v_K\}$, resulting in a face model $\{\mu, W\}$

Nearest Neighbor Recognition

- **Learning & Database Construction**
  1) Do PCA, yielding a face model $\{\mu, W\}$
  2) Construct DB of known faces with codes $y_j = W^T(x_j - \mu^T)$ for all known faces $\{x_j\}$

- **Face Recognition by NN Classification**
  1) Test face $z$ is also projected to the model $W^T(z - \mu^T) = y_z$
  2) Nearest neighbor classification of $y_z$ with $\{y_j\}$ by picking the index "i" that best match to $y_z$ according to Euclidean distance
Useful MATLAB Codes

For PCA
- Set X as a matrix with each row is a vectorized face
- \( m = \text{mean}(X) \): sample mean of X, pay attention to dim.
- \( M = \text{repmat}(\mu', 1, N) \); create a matrix by repeating a column matrix \( \mu' \) N times (M will be length of \( \mu \times N \))
- \( S = \text{cov}(X) \): covariance matrix (mean removed)
- \( [V D] = \text{eig}(S) \): eigen value decomposition of a matrix S
  - Each column of V is an eigen vector.
  - D is a diagonal matrix of eigen values.
  - Columns of V and D are corresponding to each other
- \( d = \text{diag}(D) \); vectorize the diagonal component of a matrix
- Use for-loop to get cumulative distribution of eigen values then divide it by the total variance (\( \text{sum(diag(D))} \))
- Plot(cumulative distribution of eigen values)