Heidi Gebauer (geba), Markus Roos (roor)
School of Engineering T

Zürcher Hochschule
für Angewandte Wissenschaften

PCA: Exercise Set

Remark: Throughout this exercise, we use grayscale images with the spectrum given by the input. (**NO** normalization to fit in the range $[0, 1]$ is necessary!!!)

Content: This exercise set contains 3 parts: In the first part (Task 1 - 4) you will implement the PCA algorithm, in Task 5 you will use your program for compressing images and, finally, Task 6 is about building a face detector based on PCA.

PCA Algorithm

Preparation: To load some auxiliary files run the script `loadData` (available on moodle) first.

Task 1 Implement a Python function [V] = getEigenvectors(dataMat, k) that takes as input

- a data matrix each **row** of which represents a data point, and
- an integer k,

and returns a matrix of the form $\left(v_1 \,\middle|\, v_2 \,\middle|\, \cdots \,\middle|\, v_k\right)$ with $v_1, v_2, \ldots, v_k$ representing the k first eigenvectors (assuming that they are ranked in decreasing order of eigenvalues[1]).

Hint: Use the Python commands `numpy.cov` and `scipy.linalg.eigh()`. Attention: `eigh` provides only a subset of eigenvectors with the keyword `subset_by_index=[n-k, n-1]`. However, the order is from smaller to larger eigenvalues. Therefore the order of the columns must be reversed using `numpy.flip`.

Test cases: For the dataset dataMat1, the below values are possible outcomes:

- for $k = 2$: $V = \begin{pmatrix} 0.6779 & -0.7352 \\ 0.7352 & 0.6779 \end{pmatrix}$, for $k = 1$: $V = \begin{pmatrix} 0.6779 \\ 0.7352 \end{pmatrix}$

Note: The results are not unique (multiplication with -1 gives equivalent eigenvectors).

Task 2 Implement a (one-line) Python function `m = getMean(dataMat)` which takes as input a matrix whose rows represent data points, and outputs a vector the ith component of which is the mean over the ith components of all data points.

Test case: For `dataMat1`, the result is $\begin{pmatrix} 1.8100 & 1.9100 \end{pmatrix}$.

[1] To put it precisely, the eigenvalues are sorted according to their **absolute values.**

Task 3 Implement a Python function
`vecCompressed = compressPCAVector(vec, m, vEigen)` which takes as input

- a vector representing a data point,
- a vector representing the mean-vector,
- a matrix whose columns represent eigenvectors,

and outputs a vector `vecCompressed` representing the final result of the PCA operations.

Test Cases:

- For `vecTest`, `mTest`, `vEigenTest1` the result is: $\begin{pmatrix} 0.6049 \\ 0.6031 \end{pmatrix}$
- If `vEigenTest1` is replaced with `vEigenTest2`, the result is: $\begin{pmatrix} 0.4999 \\ 0.6999 \end{pmatrix}$

Task 4

(a) In order to switch between the matrix-representation and the vector-representation of an image, implement the below auxiliary functions, each of which requires only one (very short) line of code.

- `V, dims = transformImageToVector(M)`
- `M = transformVectorToImage(V, dims)`

The additional tuple `dims` specifies the number of rows and the number of columns of `M`, respectively. **Hint:** Use the Python command `M.reshape()`.

(b) Implement (using the function of Task 3) a Python function
`ICompressed = compressPCAImage(I, nEigenvectors, V, mean)` which takes as input

- an image I,
- the number of considered eigenvectors,
- a matrix `V` whose columns represent eigenvectors,
- a matrix `mean` representing the mean face,

and outputs the image `ICompressed` resulting from applying the PCA algorithm.

Test Case: For `ITest`, `VTest`, `meanTest` and `nEigenvectors = 4`, the result is `ICompressed` $= \begin{pmatrix} 2141. & 2144. & 3267. \\ 3376. & 2569. & 4262. \end{pmatrix}$

Application I: Compressing Faces in a Database

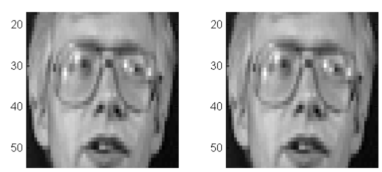
Credit: The data and also parts of the below exercise are from a lecture by Prof. Marc Pollefeys and Prof. Markus Gross

Remark You can use the auxiliary functions `loadAndScaleImage`, `displaySourceImages` and `displayCompressionResult`, to be found in `helper.py`. You do not have to modify them.

Task 5

Complete the first part of the script `task56.py` to compress images using PCA.

Remark: NEVER compute **all** eigenvectors of the covariance matrix. (There are 3808 eigenvectors, so computation time will be very long.)



Compression using 300 eigenfaces

Application II: Face Detector

Task 6

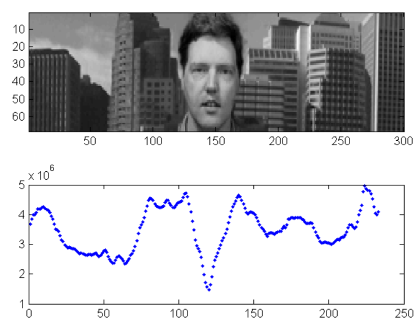
(a) Complete the Python function `faceDetector(img, d, evMatrix, mean)` which takes as input

- an image `img` whose height equals the height of each image of the database,
- an integer `d` the number of eigenvectors used of matrix `evMatrix`,
- a matrix `evMatrix` each column of which represents an eigenvector,
- a mean-face `mean`.

and proceeds as follows (c.f. the corresponding slide of the lecture for an illustration):

for $i = 1$ **to** 'end'
{
`A` := window starting at position i
`B` := compression of `A` via PCA (using `d` eigenvectors of `evMatrix` and `mean`)
`f = ssd(A, B)`
}
display original image
plot(`f`)

(b) Add a line to your script `task56.py`, which applies your face detector for $d = 50$ and `img = FaceDetection.bmp`. The eigenvectors and the mean-face `evMatrix` and `m` can be chosen in the same way as in the computations of Task 5.



Plot of the error function for each window-position.

References

[1] M. Polleyfeys and M. Gross. Visual Computing, Exercise 4. *Lecture Notes*, 2013.

[2] L. Smith. A tutorial on Principal Components Analysis. *Lecture Notes*