Exercises
Computational Intelligence Lab
SS 2010

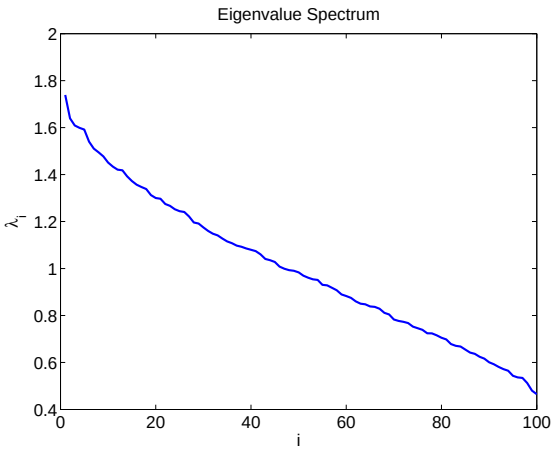
Machine Learning Laboratory
Dept. of Computer Science, ETH Zürich
Prof. Dr. Joachim M. Buhmann
Web http://ml2.inf.ethz.ch/courses/cil

Series 1, Mar 3th, 2011 (Principal Component Analysis)

Problem 1 (PCA Theory):

Principal component analysis can be used to perform image compression. Given a dataset $\mathbf{X} \in \mathbb{R}^{D \times N}$ (observations as columns), where D is the number of dimensions and N is the number of observations, a linear transformation using an orthonormal matrix is applied to make a *change of basis*, to obtain a (usually) lower-dimensional representation of the dataset denoted by $\bar{\mathbf{Z}} \in \mathbb{R}^{K \times N}$. $\bar{\mathbf{Z}}$ together with the basis (and the shift), is then used to reconstruct a compressed version the image.

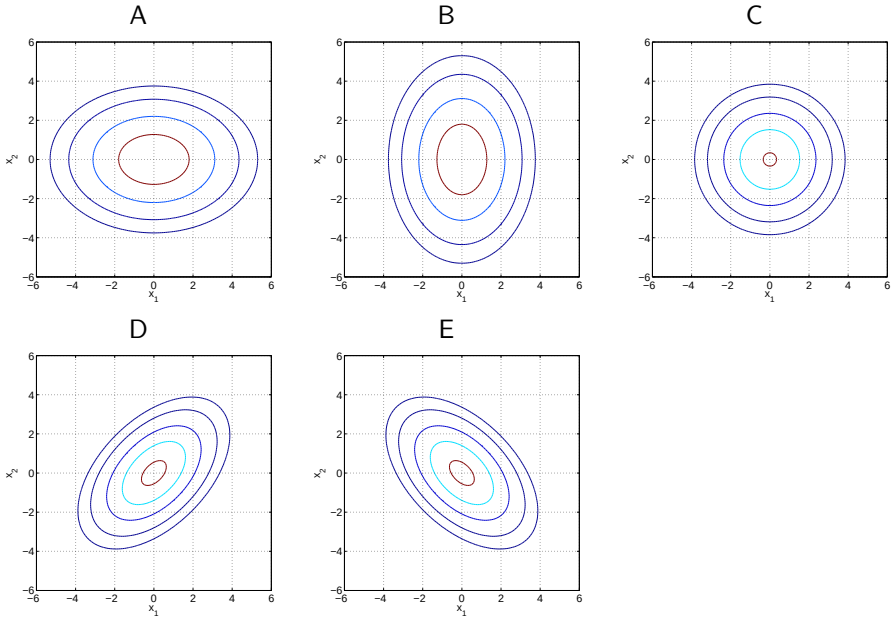
1. We begin by reviewing the steps of applying PCA to a dataset $\mathbf{X}$. Please complete each step below by providing the appropriate formula to compute the desired quantity.
   (a) Define the zero-mean dataset $\bar{\mathbf{X}}$ in terms of the original dataset $\mathbf{X}$. For this purpose, use $\mathbf{M} = [\bar{\mathbf{x}}, ..., \bar{\mathbf{x}}] \in \mathbb{R}^{D \times N}$ where $\bar{\mathbf{x}}$ is the sample mean.
   (b) Define the covariance matrix $\mathbf{\Sigma}$ in terms of the zero-mean dataset $\bar{\mathbf{X}}$.
   (c) Write down the eigen decomposition of the covariance matrix $\mathbf{\Sigma}$, in terms of the eigenvector matrix $\mathbf{U}$ (eigenvectors as columns) and the diagonal matrix of eigenvalues $\mathbf{\Lambda}$.
   (d) Define the dataset in the new basis $\bar{\mathbf{Z}}$ via a transformation of $\bar{\mathbf{X}}$. (Note: Assume we want to keep only the K dimensions of the transformed dataset and that the eigenvectors in $\mathbf{U}$ have already been sorted according to the corresponding eigenvalues, in decreasing order.)
   (e) How can the data (approximation) $\tilde{\mathbf{X}}$ be reconstructed?
2. Assume we have applied PCA to some dataset ($D = 100$). We observe the following eigenvalue spectrum of the covariance matrix of the data. (λ_i: eigenvalues)



   (a) Is the intrinsic dimensionality of this dataset low or high? Why?
   (b) Can this dataset be expressed in few dimensions with low approximation error? Why?
   (c) If yes, which dimensionality (approximately) should be chosen for the transformed dataset and why?

3. Assume you have observed 2D data $\mathbf{X} \in \mathbb{R}^{2\times N}$ (observations as columns). The first row of $\mathbf{X}$ corresponds to the first dimension x_1, the second row corresponds to x_2. For each of the three covariance matrices $\mathbf{C_X}$ below, please choose the iso-line plot (A-E) corresponding to the covariance matrix. (Note the axis labels on the figures)

1. $\begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$ Answer: ( )
2. $\begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}$ Answer: ( )
3. $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ Answer: ( )



4. PCA transforms a dataset $\mathbf{X}$ into a dataset $\mathbf{Z} = \mathbf{A}^\top\mathbf{X}$ by defining a new basis using the eigenvectors of the covariance matrix $\Sigma_\mathbf{X}$ of the dataset $\mathbf{X}$. With this particular choice of a new basis, the covariance matrix $\Sigma_\mathbf{Z}$ of the transformed dataset $\mathbf{Z}$ is *diagonalized*.

   (a) Please explain in words, why we desire the covariance matrix of the transformed dataset to be diagonal.

   (b) Show that $\Sigma_\mathbf{Z} = \mathbf{A}^\top\Sigma_\mathbf{X}\mathbf{A}$, i.e., that the covariance matrix $\Sigma_\mathbf{Z}$ of the transformed dataset can be written in terms of the covariance matrix $\Sigma_\mathbf{X}$ of the original dataset.

   (c) Show that the choice $\mathbf{A} = \mathbf{U}$ for the PCA transformation matrix, where $\mathbf{U}$ is the matrix of eigenvectors of $\Sigma_\mathbf{X} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$, actually diagonalizes the covariance matrix $\Sigma_\mathbf{Z}$ of the transformed dataset $\mathbf{Z}$. Use the fact that the inverse $\mathbf{U}^{-1} = \mathbf{U}^\top$.

Problem 2 (PCA for Image Compression):

In this assignment, we will apply principal component analysis (PCA) to image compression. We begin with first setting up the environment on your local machine for sending your compression solution to the submission system, then we go step by step through the PCA image compression pipeline.

Submission system environment setup:

1. The web page for the image compression application can be found here:

   http://cil.inf.ethz.ch/applications/compression.

2. Download the provided templates for compression and decompression functions, the evaluation script and the training data into a local folder.

3. Unzip the training images into the *same* folder.

4. Try running the evaluation script `EvaluateCompression.m` – it should output two numbers: 0 1.0001. These are the mean squared error and the compression rate of the template compression/decompression scripts.

Let's look into the `Compress.m` script now. It defines a function `I_comp = Compress(I)`, which takes an image `I` as an input and outputs a structure `I_comp` that contains a compressed version of `I`. The template provided simply stores the input image in the field of the output structure (providing no compression, whatsoever). Please place your compression code in this script.

Open the `Decompress.m` script. It defines a function `I_rec = Decompress(I_comp)`, which takes as its input the output of the `Compress` function, and returns the decompressed image. Please place your decompression code in this script.

To submit your solution to the online evaluation system, we require you to prepare a ".zip" file containing:

1. A function file called `Compress.m` which implements

   `I_comp = Compress(I),`

   which takes a color image scaled to $[0, 1]$ as an input and outputs a structure `I_comp` (the design of this structure is up to you).

2. A function file called `Decompress.m` which implements

   `I_rec = Decompress(I_comp),`

   which takes a structure returned by `Compress` as its input and returns a reconstructed image.

Your submission is evaluated according to the following criteria:

1. Compression ratio: size of the `I_comp` structure in bytes, relative to the size of original image `I`.

2. Approximation quality: mean squared error of the reconstructed image `I_rec` w.r.t. the original image `I`.

3. Run time of both the compression and decompression functions.

Working with images:

The image is represented as a matrix $M_1 \times M_2$ of pixel values. If the image has color, then there is a third dimension, representing color components – red, green and blue (RGB) – and the matrix is three dimensional now $M_1 \times M_2 \times 3$. If you want to work with each color channel independently, you can use the colon operator ":" in MATLAB to select a specific color channel: `I_red = I(:,:,1)`. To view your image, use the MATLAB function `imshow(I)`

PCA image compression:

The pipeline for a given image is:

1. Perform the feature extraction on the image.
2. Perform PCA analysis on the feature vectors.
3. Perform model selection: Which level of compression do we want?
4. Present the compressed image: Reconstruct the image and visualize the quality of the compression.

Figure 1: Original image (left) and compressed image (right).

In step-by-step fashion, we will now create all necessary modules for the compression part of the pipeline, which will provide the functionality for your version of `Compress.m`. Keep in mind that this is simply a suggested scheme. Feel free to modify the flow of your solution as much as you like as long as you keep the function calls Compress and Decompress unchanged.

1. **Feature Extraction**
   Write a function

   ```
   X = extract(I, d),
   ```

   which does the following:

   (a) Scan the image `I` and extract non-overlapping $d \times d$ pixel patches. If the image has more then one color channel do not forget to take that into account.The easiest way is to extract features in each color dimension and concatenate them afterward.

   (b) Vectorize each of these $d \times d$ blocks to create a column vector of length $D = d^2$.

   (c) Put the feature vectors together to obtain an input matrix `X` where each column represents one such square patch of the image.

   Note:

   If the image dimensions are not divisible by the size of the patch, i.e. we have $M_1 \mod d \neq 0$ or $M_2 \mod d \neq 0$, then we preprocess the image by appending the last row and column of pixels so many times until the image dimensions become divisible.

2. **PCA Analysis**
   Given the data set `X` of D features and N samples, we now find a lower dimensional projection space to approximate the data. Using the MATLAB functions `cov.m` and `eig.m`, write a function

   ```
   [mu, lambda, U] = PCAanalyse(X),
   ```

that returns the eigenvectors `U` and corresponding eigenvalues `lambda` of the PCA analysis and the mean vector `mu`.

3. **Image Compression**
   Write a function, which uses `extract(I, d)` and `PCAanalyse(X)` to compress an image:

   ```
   [I_comp] = Compress(I),
   ```

   where `I_comp` is a structure containing everything that is needed for reconstructing the original image. The size of this structure will be evaluated by the submission system.
   To implement this function you would need to set two parameters: `k, d`. The parameter `k` stands for the number of eigenvectors used. We advise you to start off by manually setting the value of these parameters. After your code runs for fixed `k and d`, you can extend your implementation by automatically selecting values which are optimal according to some criteria (also known as model selection).

4. **Image Reconstruction**
   Write a function to reconstruct the image:

   ```
   [I_rec] = Decompress(I_comp),
   ```

   `I_rec` should be an image with the same dimensions as the original image in double precision in the $[0, 1]$ interval.

5. **Model Selection (Optional)**
   We are going to represent the data with a subset of the eigenvectors. One would obviously keep the eigenvectors associated with the largest eigenvalues. But we still need to decide how many eigenvectors to keep. This can be done heuristically by plotting the eigenvalues (in descending order) and looking for the "knee" in the plot. Ultimately, you need to implement a function that does this choice automatically.

Possible extensions:

1. Try experimenting with different block sizes and number of eigenvalues used to find the best trade-off between quality and compression.
2. Different color channels are quite similar. You might try to exploit this to obtain better compression.
3. For the largest eigenvalues, visualize the associated image segment. For this you have to convert the eigenvector back to a $d \times d$ block. Look at the plot of eigenvalues and try to come up with automatic algorithm, that selects the number of eigenvalues to be used for compression.