Exercise 6: Reduction of dimensionality and recognition

Machine perception

2018/2019

Create a folder `exercise6` that you will use during this exercise. Unpack the content of the `exercise6.zip` that you can download from the course webpage to the folder. Save the solutions for the assignments as the *Matlab/Octave* scripts to `exercise6` folder. In order to complete the exercise you have to present these files to the teaching assistant. Some assignments contain questions that require sketching, writing or manual calculation. Write these answers down and bring them to the presentation as well. The tasks that are marked with ★ are optional. Without completing them you can get at most 75 points for the exercise (the total number of points is 100 and results in grade 10). Each optional exercise has the amout of additional points written next to it.

Introduction

In the following tasks we will test the Principal Component Analysis (PCA) method, which has been presented at the lectures. First, we discuss a simple 2D example of the direct PCA method, which is easier to visualize. You will solve this example on paper, but at the same time visualize the data in Matlab/Octave. Next, you'll experiment with the *dual* PCA method in the case of compact face description and try out some nice properties of such a description. More about the theory of PCA and its applications can be found in [1] (section 22.3.1, pages 507 to 515).

Assignment 1: Direct PCA method

The primary purpose of the PCA method is reduction of dimensionality. This means that we want to find such a linear transformation of the input data that will describe the data in a low dimensional orthogonal space (which we call subspace) while losing minimal amount of information needed to reconstruct the data. Subspace is defined by its basis vectors. Let us first repeat the direct calculation of the basis vectors with the PCA method. Here we assume that the input data is structured as a column vector $\mathbf{x}$ with size $m \times 1$ (they have m cells):

In the algorithm we can see that in first steps of the PCA method we compute the mean value and the covariance matrix of the data. These are the two parameters that define a Gaussian distribution. As we will later on refer to this distribution we should give it a little more attention. We have already introduced Gaussian distribution in Exercise 2, where we have used a special case of such distribution to construct a Gaussian kernel. In that case the distribution had a mean value of zero and a diagonal covariance matrix with

Algorithm 1 : The direct PCA algorithm.

1: Build a matrix $\mathbf{X}$ of size $m \times N$ so that we combine N input entries as a sequence: $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]$

2: Calculate the mean value of the data: $\mu = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i$

3: Center the data: $\mathbf{X}_d = [\mathbf{x}_1 - \mu, \mathbf{x}_2 - \mu, \ldots, \mathbf{x}_N - \mu]$

4: Compute a covariance matrix: $\mathbf{C} = \frac{1}{N-1} \sum_{i=1}^{N} (\mathbf{x_i} - \mu)(\mathbf{x_i} - \mu)^T = \frac{1}{N-1} \mathbf{X}_d \mathbf{X}_d^T$

5: Compute SVD of the covariance matrix: $\mathbf{C} = \mathbf{USV}^T$ The matrix $\mathbf{U}$ is of size $m \times m$, its columns represent eigenvectors of the matrix $\mathbf{C}$. The corresponding eigenvalues are saved in a diagonal matrix $\mathbf{S}$ and are sorted in a descending order. The columns in $\mathbf{U}$ therefore represent an orthogonal basis (subspace) while the eigenvalues give us the level of scattering of the data along the corresponding eigenvector.

6: A new input entry $\mathbf{x}_q$ is projected in the PCA space (subspace) by first centering it using μ, and then multiply it with the matrix of eigenvectors: $\mathbf{y} = \mathbf{U}^T(\mathbf{x}_q - \mu)$.

7: The transformation from the subspace to the original space is done using equation: $\mathbf{x}_q = \mathbf{Uy} + \mu$.

equal covariance along the x and y axis. The general formula for a Gaussian distribution is, however

$$p(\mathbf{x}) = \frac{1}{\sqrt{2\pi}^d |\mathbf{C}|^{\frac{1}{2}}} \exp(-\frac{1}{2}(\mathbf{x} - \mu)^T \mathbf{C}^{-1}(\mathbf{x} - \mu)). \tag{1}$$

Image 1(a) shows an example of a Gaussian distribution where the values on the y axis display the probability of $p(\mathbf{x})$. Image 1(b) shows an alternative illustration, where we can see a border at which the ellipse contains the 68% of all probability density. We will use the former display in the assignment to visualize eigenvalues and the components. Points used to compute the mean value and the covariance used to draw a Gaussian distribution are shown in the right image.

How to calculate eigenvectors and eigenvalues analytically?

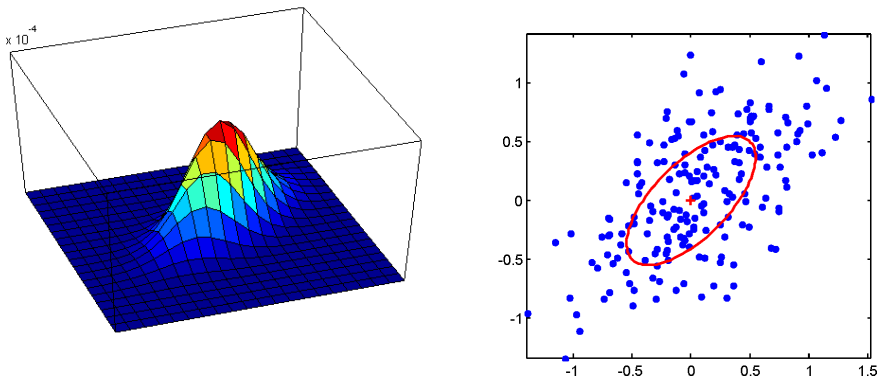
Let us briefly repeat how to calculate eigenvectors and eigenvalues of a square matrix $\mathbf{C}$ with size $m \times m$ analytically. Consider a matrix $\mathbf{Z}$, that we get if we multiply a unit matrix of size $m \times m$ with a constant value, e.g., $\mathbf{Z} = \lambda \mathbf{I}$. If we subtract this matrix from $\mathbf{C}$ and we get a determinant 0, we have transformed $\mathbf{C}$ into a singular matrix:

$$|\mathbf{C} - \lambda \mathbf{I}| \equiv 0. \tag{2}$$

The values λ that satisfy the equation above are called *eigenvalues* of the matrix $\mathbf{C}$. For $m \times m$ matrix we can compute m eigenvalues λ_i. A product of eigenvalues equals the determinant of the matrix: $|\mathbf{A}| = \prod_{i=1}^{m} \lambda_i$. Each eigenvalue λ_i corresponds to a *eigenvector* $\mathbf{x}_i$. Eigenvectors are mutually perpendicular (dot product of two eigenvectors is 0).

For eigenvalue λ_i we compute eigenvector $\mathbf{x}_i$ of size $1 \times m$ by setting up the following equation:

$$(\mathbf{C} - \lambda_i \mathbf{I})\mathbf{x}_i \equiv 0. \tag{3}$$



(a) (b)

Figure 1: The left image displays a tabulated example of a 2D Gaussian distribution, the right image shows an example of the distribution where we only display values of $\mathbf{x}$ with a constant probability.

The solution of the equation gives us elements of the eigenvector up to the level of scale. That means that the equation can be solved by an infinite number of vectors that only differ in scale. In practice we pick value 1 for the first element and then calculate other elements of the vector. In the end we change the scale of the vector so that its length becomes 1 (unit vector). If we assume that vector $\mathbf{x}_i$ solves the equation above. Unit eigenvector for λ_i is defined as $\mathbf{e}_i = \mathbf{x}_i / ||\mathbf{x}_i||$.

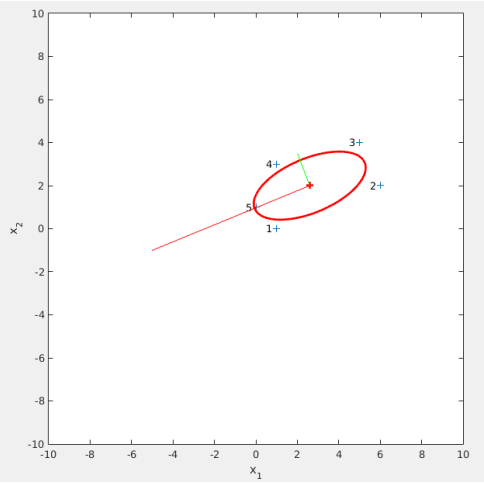
For a longer explanation on how to compute eigenvectors as well as some examples, check this link. Solve the following assignment by hand for practice: You are given four points $A(3,4)$, $B(3,6)$, $C(7,6)$ and $D(6,4)$. Calculate the eigenvectors and eigenvalues for the given set of points.

A simple example of a direct PCA

(a) Run script `direct_pca_demo`. The script will load and visualize 2D data from the file `points.txt`. Extend the function by following the Algorithm 1 to compute PCA (eigenvectors and eigenvalues) of the data from `points.txt`. To compute the decomposition of the covariance matrix $\mathbf{C}$ use the singular value decomposition `[U, S, V] = svd(C)`. Draw the covariance matrix and the mean value on the figure that has been used to draw the points. To visualize the distribution use function `draw_gauss2d(Mu, C, 'r', 1)`.

(b) Matrix $\mathbf{U}$ is composed of eigenvectors that represent the basis of our subspace. On the figure that you have used in the previous step also draw the eigenvectors of the matrix $\mathbf{C}$ with the origin in the mean value of the data μ. Both eigenvectors have a length of 1. For a better visualization of eigenvectors multiply each vector with the corresponding eigenvalue[1], before you translate them to the origin μ. Draw the first eigenvector with green color and the second one with red. Comment on what you notice.

[1]Interesting fact: Eigenvalue tells us the variance of the data along its eigenvector. If you therefore multiply the eigenvector with a square root of the eigenvalue you will visualize the standard deviation along the eigenvector.



(c) At the lectures you have heard that the eigenvalues represent the reconstruction error in case if we discard the corresponding eigenvector. Plot the cumulative graph of the eigenvalues and normalize it so that the largest value will be 1. From the graph determine how many percents of the reconstruction information we still have if we retain only the first eigenvector and discard the second one. Or, to say it otherwise, *what percent of the variance* is explained just by using the first eigenvector? For the calculation of cumulative values use function `cumsum`.

(d) Now we will remove the *direction of the lowest variance* from the input data – we will project the data in the subspace of the first eigenvector. We do this by first transforming the data in the PCA space. Then we set to 0 the components of the corresponding eigenvectors that we want to remove. Then we transform the vectors back in the original space. Draw the data, modified this way on the image from the task (c) above. Use function `draw_reconstructions` that takes as the input the original and the reconstructed data and draws the links between the corresponding points for better visualization of the projection. What do you notice? Where is (geometrically) the data projected to?

(e) For point $q_{\text{point}} = [3, 6]^T$ compute the closest point among the input data using Euclidean distance. Which point is the closest? Now, project all the input points as well as q_{point} in the subspace where you remove the variation in the direction of the second eigenvector. Which input point is the closest to the q_{point} in this subspace? Comment on the result. Visualize the projection (reconstruction) of the point q_{point} on the same figure used in the previous tasks.

Assignment 2: The dual PCA method

If we have to analyze datasets, where the dimensionality of the data is very high and it is in fact much greater than the number of the elements in the input data, the direct method of computing the eigenvectors that we used in the previous section is not suitable. If we have, e.g. 10000-dimensional data, we will get a covariance matrix of size 10000×10000. At these sizes of the matrices we easily hit the limitations of the computer memory. As the number of data samples N is much lower than the dimensionality we can use the *dual PCA method*. This method only differs from the direct PCA method in the steps four and five (compare Algorithm 1 with Algorithm 2).

(a) For our needs it is only necessary to correctly calculate eigenvectors and eigenvalues up to the scale factor. Therefore implement the dual method according to the Algorithm 2 and test it using the data from `points.txt`. Verify whether you get the same eigenvectors (matrix $\mathbf{U}$) as with the Algorithm 1. The first two eigenvectors should be the same as with the Algorithm 1. The Algorithm 2 gives you a larger matrix $\mathbf{U}$, however all eigenvectors but the first two equal to zero. For the projection from and to the subspace you can leave the matrix $\mathbf{U}$ as it is.

(b) As a final test project the data from the previous assignment to the PCA space using matrix $\mathbf{U}$, and then project the data back again in the original space. If you have implemented the method correctly, you will get the original data (up to the numerical error).

Algorithm 2 : The dual PCA algorithm.

1: Build a matrix $\mathbf{X}$ of size $m \times N$ so that we combine N input entries as a sequence: $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]$
2: Calculate the mean value of the data: $\mu = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i$
3: Center the data: $\mathbf{X}_d = [\mathbf{x}_1 - \mu, \mathbf{x}_2 - \mu, \ldots, \mathbf{x}_N - \mu]$
4: Compute a dual covariance matrix: $\tilde{\mathbf{C}} = \frac{1}{m-1} \mathbf{X}_d^T \mathbf{X}_d$.
5: Compute singular value decomposition (SVD) of the dual covariance matrix, $\tilde{\mathbf{C}} = \tilde{\mathbf{U}} \tilde{\mathbf{S}} \tilde{\mathbf{V}}^T$. Then compute the basis of the eigenvector space as: $\mathbf{U} = \mathbf{X}_d \tilde{\mathbf{U}} ((m-1)\tilde{\mathbf{S}})^{-1/2}$. Note: it is expected that only the first few of the m or N (the lower one) columns of the eigenvectors will differ from 0.
6: A new input entry $\mathbf{x}_q$ is projected in the PCA space (subspace) by first centering it using μ, and then multiply it with the matrix of eigenvectors: $\mathbf{y} = \mathbf{U}^T(\mathbf{x}_q - \mu)$.
7: The transformation from the subspace to the original space is done using equation: $\mathbf{x}_q = \mathbf{U}\mathbf{y} + \mu$.

Assignment 3: Image decomposition examples

In the following assignment we will test the implementation of the dual PCA method on a computer vision problem. In the data that you have received along this instructions there are three series of images available. Each series contains 64 images of a face under different lightning conditions. Your task is to analyze each series using the PCA method.

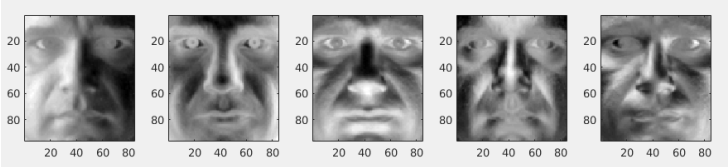
(a) **Data prepraration**: The first thing that we have to do is to formulate the problem in a way that is compatible with the PCA method. As you have found out during previous tasks the PCA method operates on points in space. A grayscale image of size $m \times n$ can be represented as a point in a mn-dimensional space if we reshape the image into a vector. Write a function that reads images from one of the series (the number of series is given to the function as an input parameter) in the memory, transforms them into grayscale if needed, reshapes them (using the `reshape` function) into column vectors and returns a matrix of stacked vectors of size $64 \times mn$. In the following task use the first series as a reference series, however, you can test your algorithms with the other two series as well.

(b) **Using dual PCA**: Use your dual PCA implementation with the vectors of images. Write the method in a form of a function that gets a matrix of vertices as an input and returns eigenvectors of the PCA subspace and the mean value of the input data.

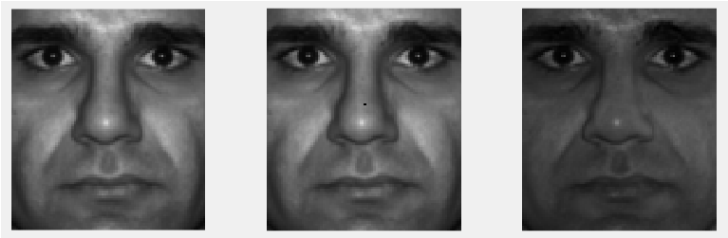
In the step 5 of the Algorithm 2 be careful when computing an inverse of the $\tilde{\mathbf{S}}$ as some of the eigenvalues can be very close to 0. Division by zero can cause numerical errors when computing a matrix inverse. You have to take into account that the matrix $\tilde{\mathbf{S}}$ is a diagonal matrix and must therefore have non-zero diagonal elements. One way of solving this numerical problem is that we add a very small constant value to the diagonal elements, e.g. 10^{-15}. A numerically stable calculation of the inverse of $\tilde{\mathbf{S}}$ can be implemented as:

```
s = diag(S) + 1e-15; % Extract the diagonal as a vector and add a small constant
Si = diag(1 ./ sqrt(s * (m - 1))); % Compute the inverse and construct a diagonal matrix
```

Transform the first five eigenvectors using the `reshape` function back into a matrix and display them as images (it is recommended that you use the `imagesc` function to display them). How do you explain the result?

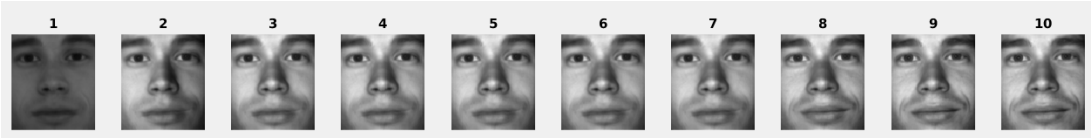


Project the first image from the series to the PCA space and then back again. Is the result the same? What do you notice when you change one dimension of the vector in the image space (e.g. component with index 4074) to 0 and display the image? Repeat a similar procedure for a vector in the PCA space (project image in the PCA space, change one of the first five components to zero and project the image back in the image space and display it as an image). What is the difference? How many pixels are changed by the first operation and how many by the second[2]?



(c) **Effect of the number of components on the reconstruction:** Take image 24 and project it into the PCA space. Then change the vector in the PCA space by retaining only the first 32 first components and setting the remaining components to 0. Project the resulting vector back to the image space and display it as an image. Repeat the procedure for the first 16, 8, 4, 2, and one eigenvector. Display rhe resulting vectors together on one figure. What do you notice?

[2]visualize the difference by subtracting the original image from the reprojected one using `imagesc`.



(d) ★ (5 points) **Informativeness of each component**: Use the second series of images for this task. Take the average photo that you compute based on all images in the series[3]. Project the image in the PCA space. Then change the values of the second and third component in the projected vector (add or remove 2) and project the vector back in the image space. Display the original image and the modified image. What do you notice? After that implement this modification in a loop as well. The second component should be changed according to formula $x \cdot sin(x)$ and the third component according to formula $x \cdot cos(x)$, where x is a scaling value (by default try value 5, but increase the value if you observe no visible change in the animation).

Vary values of x from -20 to 20 in 500 steps. For each step project the changed vector back into the original space and display the image in the same figure as an animation. Display each modified image for 0.1 seconds in the same figure, then show the next image. This way you will get the effect of an animation and you will be able to notice the effect of each component on the entire image more easily. Take the bottom code as a start when writing your animation script:

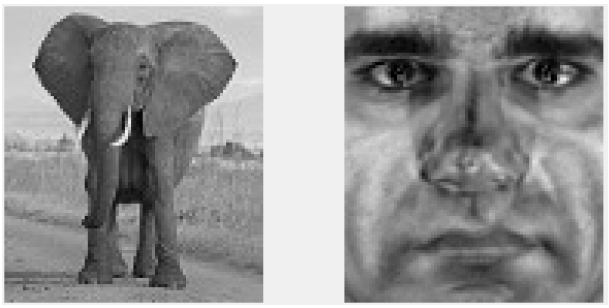
```
% TODO: compute PCA

for x = linspace(-20, 20, 500)

    % TODO: compute the projection, reshape it to image and store it to the variable image

    figure(1); imshow(uint8(image)) ;
    pause(0.1);
end;
```

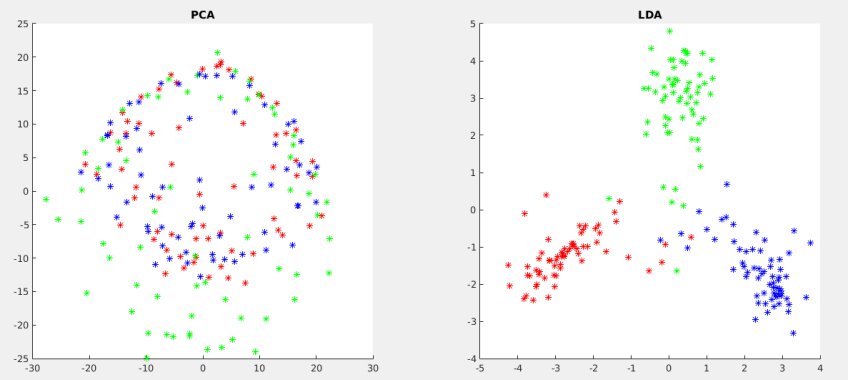
(e) ★ (5 points) **Reconstruction of foregin image**: In this task inspect the effect that the translation to PCA space has on an image that is not similar to the images that were used to build the PCA space. First compute the PCA space for the first face dataset. Then load image `elephant.jpg`, project it to PCA space and then back to image space. Inspect the reconstruction and write down your comments.



[3]The value of each pixel equals to the average value of the corresponding pixels in all images. In case of image vectors we are dealing simply with their average value.

(f) ★ (15 points) **Recognition with subspace**: The properties from the previous task can be used in a simple object recognition scenario. Assemble a collection of at least 10 images of your face (align them in terms of size and eye location) with varying illumination and facial expressions. Construct a PCA subspace and create an interface that captures images from a webcam[4], visualizes frames, cuts out a sub-image (of appropriate size) and project it to the subspace and then back to the image space. Then determine if the sample contains your face or not based on the similarity of the original and reconstructed sample (using L2 norm or some other distance measure with appropriate threshold). *Note*: do not use images from the provided dataset for this task as you will not be able to verify the scenario with a webcam. The only way to complete this exercise is by using a custom collection of your own images.

(g) ★ (10 points) **Linear discriminant analysis**: Based on the instructions from the lecture slides (*Subspace-based recognition*) implement the LDA algorithm and apply it to our face database. Make sure that you also address the problem of singularities in high-dimensional data using appropriate pre-processing (also described in the lecture slides). Visualize all the image samples in the 2-D PCA subspace and the LDA subspace with different colors for each class to demonstrate that the LDA subspace better separates classes.



References

[1] D. A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Prentice Hall, 2002.

[4] Webcam acquisition modules are available for Matlab as well as Octave.