Machine Learning Programming
Assignment 1: Principal Component Analysis

Professors: Baptiste Busch, Aude Billard
Assistants: Lukas Huber, Laila El Hamamsy,
Matthieu Dujany and Victor Faraut
Contacts:
baptiste.busch@epfl.ch, aude.billard@epfl.ch
lukas.huber@epfl.ch, laila.elhamamsy@epfl.ch, matthieu.dujany@epfl.ch, victor.faraut@epfl.ch

Winter Semester 2018

Introduction

This series of practicals focus on the development of machine learning algorithms introduced in the Applied Machine Learning course. In this first practical, you will code the Principal Component Analysis (PCA) algorithm in Matlab. You will then use the code to reduce the dimensionality the Fisher-Iris dataset, a real-world 9d dataset and compress a single image.

Submission Instructions

Deadline: October 16, 2018 @ 6pm. Assignments must be turned in by the deadline. `1pt` will be removed for each day late (a day late starts one hour after the deadline).
Procedure: From the course Moodle webpage, the student should download and extract the .zip file named `TP1-PCA-Assignment.zip` which contains the following files:

Part 1 - Algorithm	Part 2 - Applications
`compute_pca.m`	`plot_eigenvalues.m`
`project_pca.m`	`explained_variance.m`
`reconstruct_pca.m`	`test_pca_9d.m`
`reconstruction_error.m`	`compress_image.m`
`test_pca_iris.m`	`compression_rate.m`
	`reconstruct_image.m`
	`test_pca_compression.m`

As well as `TP1-PCA-Dataset.zip` which contains the datasets required to test your functions.

Assignment Instructions

The assignment consists of implementing the blue colored MATLAB functions from scratch. These functions can be tested with the `test_*.m` scripts. These testing scripts depend on `ML_toolbox`, which is already installed on the Virtual Machines. In case you want to work on your own computer you will have to download it from: `https://github.com/epfl-lasa/ML_toolbox` and change the path in each scripts by modifying the following line.

```
>> addpath(genpath('path_to_ML_toolbox'))
```

Once you have tested your functions, you can submit them as a .zip file with the name: `TP1-PCA-Assignment-SCIPER.zip` on the submission link in the Moodle webpage. Your submission archive should contain ONLY the following:

```
🟥 TP1-PCA-Assignment-SCIPER
 └─🟦 Part1
 └─🟦 Part2
```

DO NOT upload `ML_toolbox`, the `check_utils` directory or the `Datasets` directory.

1 Part 1: Implementing Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a well-known dimensionality reduction technique which projects the data onto a new basis of smaller dimensions. It is used as:

- Pre-processing step for clustering or classification algorithms to reduce the dimensionality and computational costs.
- Compression for data storage and retrieval.
- Feature extraction.

Given a training dataset $\mathbf{X} \in \mathbb{R}^{N\times M}$ composed of M datapoints with N-dimensions each; i.e. $\mathbf{X} = \{\mathbf{x}^1, \mathbf{x}^2, ..., \mathbf{x}^M\}$ where $\mathbf{x}^i \in \mathbb{R}^N$ we would like to find a lower-dimensional projection $\mathbf{Y} \in \mathbb{R}^{p\times M}$ (i.e. $\mathbf{Y} = \{\mathbf{y}^1, \mathbf{y}^2, , ..., \mathbf{y}^M\}$ where $\mathbf{y}^i \in \mathbb{R}^p$) of $\mathbf{X}$ through a linear map $A : \mathbf{x} \in \mathbb{R}^N \to \mathbf{y} \in \mathbb{R}^{p\leq N}$ given by

$$\mathbf{y} = A\mathbf{x}$$

where the projection matrix $A \in \mathbb{R}^{p\times N}$ is found through Principal Component Analysis. For a thorough description of PCA, its application and derivation, refer to the PCA slides from the Applied Machine Learning Course.

PCA Algorithm

By computing the covariance matrix of the training data $C = \mathbb{E}\{\mathbf{X}\mathbf{X}^\mathbf{T}\}$ and extracting its eigenvalue decomposition $C = V\Lambda V^T$, the projection matrix A is constructed as $A = V^T$ where $V = [e^1, \dots, e^N]$. To reduce dimensionality, one then chooses a sub-set of p eigenvectors e^i from V.

1.1 Demean the Data

We will follow the steps of the PCA algorithm shown on slide 62 of the class, starting with the centering step,

$$\mathbf{x} \to \mathbf{x} - \mathbb{E}\{\mathbf{X}\} \tag{1}$$

where the expectation of a random variable $\mathbb{E}\{\mathbf{X}\}$ is in fact it's mean. The substraction of the mean is necessary for the eigenvalue decomposition of the covariance matrix to yield the expected result. Hence, the $\mathbb{E}\{\mathbf{X}\}$ of a multi-dimensional vector is the mean vector of all dimensions $\mu_\mathbf{X} = [\mu_1; \mu_2; \dots; \mu_N]$. For each j-th dimension, its mean μ_j can be computed as follows:

$$\mu_j = \frac{1}{M}\sum_{i=1}^{M} x_j^i.$$

`Implementation Hint:  Useful functions mean(), repmat() or bsxfun().`

1.2 Covariance Matrix Computation

Once the data is demeaned, the next step in PCA is to compute the Covariance matrix $C = \mathbb{E}\{\mathbf{X}\mathbf{X}^{\mathbf{T}}\}$ of our centered (zero-mean) data, as follows:

$$C = \frac{1}{M}\sum_{i=1}^{M}(\mathbf{x}^i)(\mathbf{x}^i)^T \tag{2}$$

`Implementation Hint:  2 can be written in matrix formulation as follows:`

$$C = \frac{1}{M}XX^T$$

Although (2) is the definition of covariance matrix, in your MATLAB function, you should normalize by $M-1$; i.e. $C = \frac{1}{M-1}XX^T$. The reason C is normalized by $M-1$ instead of M is due to the fact that the true $\mathbb{E}\{\mathbf{X}\}$ is not known. According to Bessel's correction, one must use $M-1$ in order to have an unbiased estimate, because we use the sample mean $\mu_{\mathbf{X}}$ instead of the true mean. Refer to Applied Multivariate Statistical Analysis book for further reading on this subject.

1.3 Eigenvalue Value Decomposition

The principal components of our dataset can then be computed by extracting the eigenvalues and eigenvectors of the covariance matrix C. This is generally done by solving the following equations:

$$Ce^i = \lambda_i e^i \implies \det(C - \lambda I) = 0$$

which leads to the solution,

$$C = V\Lambda V^T \tag{3}$$

where $V \in \mathbb{R}^{N\times N}$ is the matrix composed of the eigenvectors e^i on each column and $\Lambda \in \mathbb{R}^{M\times M}$ is a diagonal matrix of eigenvalues λ_i.
`Implementation Hint:  Useful functions eig() or svd().`

TASK 1: Implement compute_pca.m function (3pts)
We must ensure that $V = [e^1, e^2, \ldots, e^p] \quad \text{for} \quad \lambda_1 \geq \lambda_2 \cdots \geq \lambda_p$.

```
function [V, L, Mu] = compute_pca(X)

% 1.1 Data Centering
Mu = ...;
X  =  ...; % Equation 1

% 1.2 Covariance Matrix Computation
C  = ...; % Equation 2

% 1.3 EigenValue Decomposition
[V,L] = ... ; % Equation 3

end
```

Test Implementation

At this point, we can test that we have implemented `compute_pca.m` function correctly, by running the **first two** code blocks in the MATLAB script `test_pca_iris.m`. This will load the

Iris dataset directly from Matlab. As a reminder, the Iris dataset contains the four measurements of 150 Iris flowers as illustrated in Fig. 1. Features are stored in the *meas* array of dimensions 150×4.

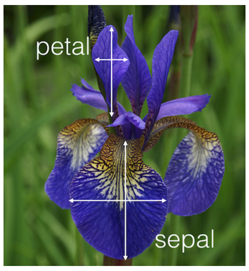

Figure 1: An Iris flower with the four type of measurements

When running the script, it will compare the output of your function $[V, \Lambda, \mu_{\mathbf{x}}]$ to the PCA function from `ML_toolbox`. It will also plot the scatter matrix of the original data as shown in Fig. 2.

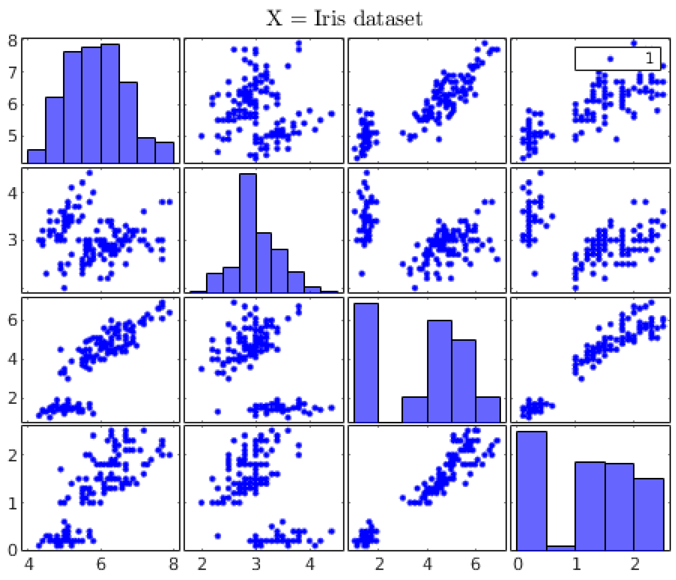

Figure 2: The Iris dataset

If your implementation is correct, you should see the following messages in your MATLAB command line window:

```
    [Test1] Error in Mean Computation: 0.00
    [Test2] Error in Eigenvalue Computation: 0.00
    [Test3] Error in Eigenvector Computation: 0.00
```

1.4 Projection to Lower Dimensional Space

One then chooses the number of p components/eigenvectors to keep and selects the first p columns of V to generate the projection matrix A_p

$$A_p = [e^1, e^2, \ldots, e^p]^T \qquad \text{for} \qquad \lambda_1 \geq \lambda_1 \cdots \geq \lambda_p \tag{4}$$

One can then project the zero-mean dataset to the lower-dimensional space as follows:

$$Y = A_p X \tag{5}$$

TASK 2: Implement project_pca.m function (2pts)
Recall that the projection matrix A_p was computed with zero-mean data, hence X in (5) must be the zero-mean dataset.

```
function [A_p, Y] = project_pca(X, Mu, V, p )

% 1.4 Projection to Lower Dimensional Space
A_p = ... ; % Equation 4

X = ... ; % Demean data if necessary
Y = ... ; % Equation 5
```

Test Implementation

To test the `project_pca.m` function, run the **third** code block in the MATLAB script `test_pca_iris.m`. The expected result of the projection code block, with $p = [2, 3]$, is shown in Figure 3 we can see how the data is distributed on each eigenvector (histograms) and how it can be decorrelated if we project on two eigenvectors (scatter plots).

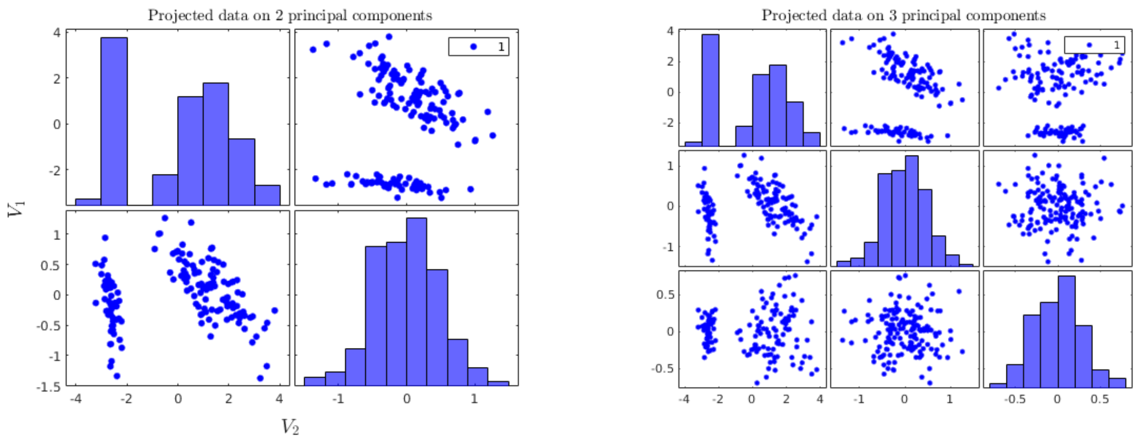

Figure 3: Scatter matrix of dataset projected on 2 (left figure) and 3 (right figure) eigenvectors.

If your histograms and projected data seem to be mirrored to those of Figure 3, this is a result of the opposite signs of V computed from `eig()` or `svd()`, and should yield the same reconstruction. For different MATLAB versions the color and bins of the histograms might change, this does not affect your result.

1.5 Reconstruction from Lower Dimensional Space

Now that we know that `compute_pca.m` and `project_pca.m` functions are correct, we can implement the reconstruction function. This can be done by reconstructing a lossy version $\hat{\mathbf{X}}$ of the original dataset $\mathbf{X}$ through mean-square projection as follows:

$$\hat{\mathbf{X}} = A_p^{-1}\mathbf{Y} + \mathbb{E}\{\mathbf{X}\}. \tag{6}$$

TASK 3: Implement reconstruct_pca.m function (1pt)

```matlab
function [X_hat] = reconstruct_pca(Y, A_p, Mu)
% 1.5 Reconstruct from Lower-Dimensional Space
X_hat = ...; %Equation 6
```

To estimate how much information was lost from $\mathbf{X} \to \hat{\mathbf{X}}$, we can compute the reconstruction error as follows:

$$e_{rec} = ||\mathbf{X} - \hat{\mathbf{X}}||_2 \tag{7}$$

TASK 4: Implement reconstruction_error.m function (1pt)

```matlab
function [e_rec] = reconstruction_error(X, X_hat)
e_rec = ...; %Equation 7
```

`Implementation Hint: Useful function norm().`

Test Implementation

The **last** code block in `test_pca_iris.m` tests your reconstruction functions. By running, it you should get the result in Figure 4, which is computed from an A_p with $p = 3$ and yields a $e_{rec} = 1.884524$. The following message should appear in your MATLAB command line window:

```
Reconstruction Error with p=1 is 1.884524
```

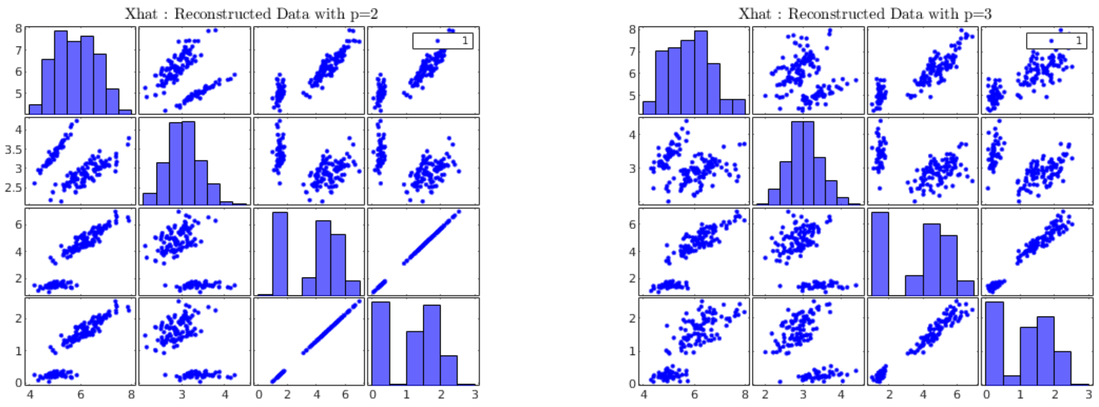

Figure 4: Reconstruction with $p = 2$ ($e_{rec} = 3.413681$) and $p = 3$ ($e_{rec} = 1.884524$) components.

Although the error is relatively large for 3 components, it is still a viable compact representation of the data. In the next part of the assignment, we will implement methods for selecting the best p for your dataset and application.

2 Part 2: Application of PCA to High Dimensional Data

2.1 PCA for Pre-Processing, choosing p

In this exercise we will use the Breast-Cancer-Wisconsin (Diagnostic) Dataset[1] which is composed of $M = 698$ datapoints of $N = 9$ dimensions, each corresponding to cell nucleus features

[1]The Breast-Cancer-Wisconsin (Diagnostic) Dataset can be found in the UCI Machine Learning Repository: `https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29`

in the range of $[1, 10]$. The datapoints belong to two classes $y \in \{\text{benign}, \text{malignant}\}$ (See Figure 5). With such datasets, one would like to reduce the dimensionality to possibly improve classification by finding correlations in the features. Another reason is to reduce computational costs. To achieve this, one must find the optimal p (the number of principal components to use), which can be the determined in two ways:

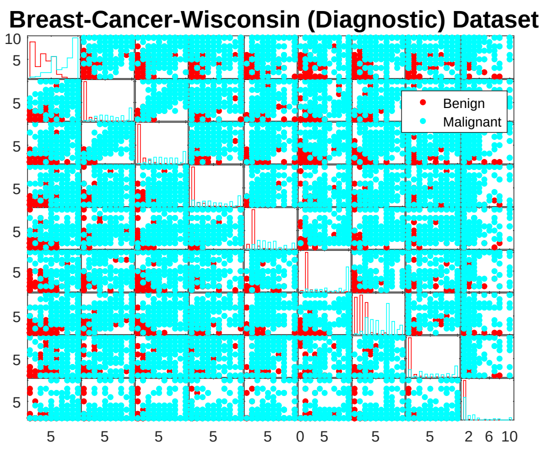

Figure 5: Scatter Matrix of Breast Cancer Wisconsin Dataset on Original Dimensions.

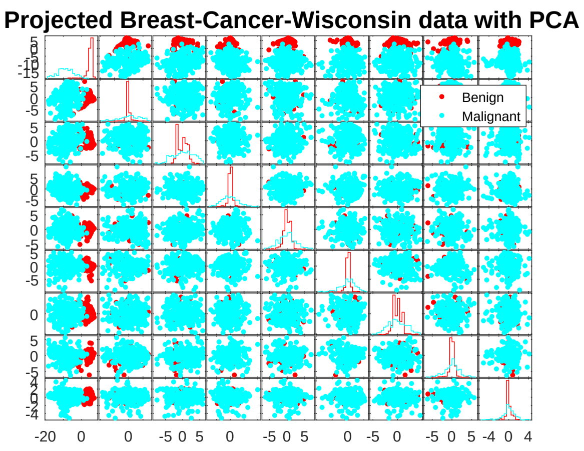

Figure 6: Scatter matrix of dataset projected to each eigenvector.

1) Eigenvalue Analysis

The behavior of the eigenvalues can be used to determine the optimal number of components. When the values corresponding to each principal component (eigenvalues) become constant and close to 0, it could mean that the remaining components are possibly contaminated with noise and can therefore be considered irrelevant. This can be analyzed by extracting the diagonal values λ_i of Λ from (3) and plotting them wrt. their corresponding eigenvector index.

TASK 5: Implement plot_eigenvalues function (1pt)

```
function [lambda] = plot_eigenvalues(L)
lambda = ...; % Column-vector of eigenvalues
```

Test Implementation

We can generate both Figure 5, corresponding to the original dataset, and Figure 6, corresponding to the projected dataset with the functions implemented in Part 1, by running the **first two** code blocks in the MATLAB script `test_pca_9d.m`. For this script to run properly, modify `ml_toolbox_path` at the beginning of the script to the root directory of `ML_toolbox`.

To test your `plot_eigenvalues` function, run the **third** code block in the MATLAB script `test_pca_9d.m`. The expected output is `lambda` a column-vector filled with the eigenvalues λ_i and Figure 7. As we can see in Figure 7, from component (eigenvector) number 2/3 and up, the eigenvalues are negligible compared to the first one, indicating that setting p to either 2 or 3 could be enough to represent the dataset correctly. From Figure 6 one can see that when $p = 2$ the datapoint form the two classes seem to be separated nicely, with some overlapping points. This can be better visualized in Figure 8 when $p = 3$. In fact, if we analyze the projection on the first eigenvector (in Figure 6) we can see this nice separation already. This can indicate that the first eigenvector explains much of the variance of the dataset and that it also encapsulates the correlations between most of the dimensions.

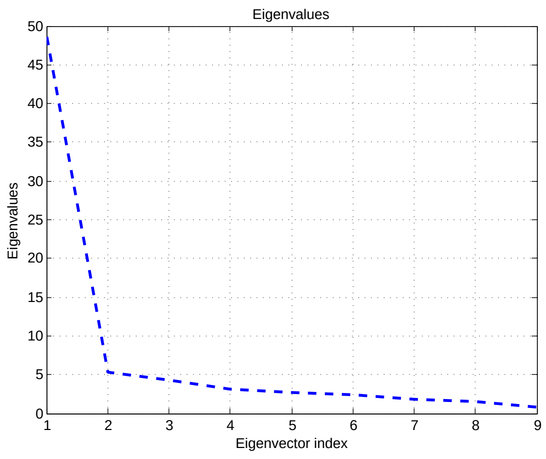

Figure 7: Plot of Eigenvalues

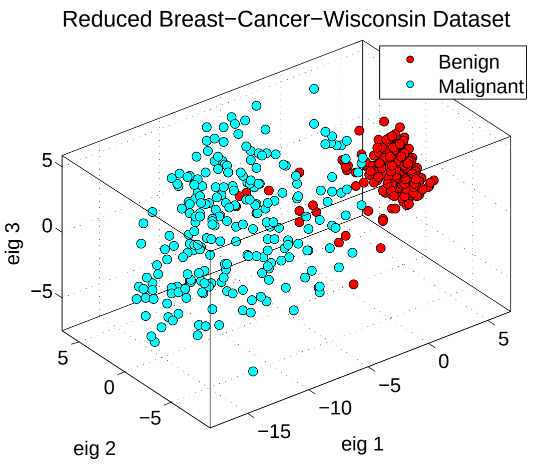

Figure 8: Projected Dataset with p=3

2) Percentage of Variance Explained

Instead of searching for the minimum number of dimensions, another approach is to search for the optimal number of dimensions to keep in order to explain a certain amount of variance of the dataset. The eigenvalues $\Lambda = [\lambda_1, \dots, \lambda_N]$ give a measure of the variance of the distribution of X on each projection. If one wants to reduce the dimensionality of the dataset, and at the same time keep a reasonable amount of information of the data, one can find the appropriate p that yields the desired percentage of explained variance σ, by normalizing the eigenvalues as follows:

$$\lambda_i^{var} = \frac{\lambda_i}{\sum_{j=1}^{N} \lambda_j}. \tag{8}$$

This corresponds to the percentage of the dataset covered by the i-th projection. One then computes a vector of the cumulative sum of the set of normalized eigenvalues

$$\lambda_{\mathbf{cum}} = [\lambda_1^{var}, \lambda_2^{var} + \lambda_1^{var}, \dots, \lambda_N^{var} + \lambda_{N-1}^{var}]. \tag{9}$$

One can then use $\lambda_{\mathbf{cum}}$ to estimate the optimal p for a desired percentage of explained variance σ as

$$p = \underset{k}{\operatorname{argmin}}(\lambda_{\mathbf{cum,k}}) \quad \text{with} \quad \lambda_{\mathbf{cum,k}} > \sigma. \tag{10}$$

TASK 6: Implement explained_variance function (3pts)

```
function [exp_var, cum_var, p] = explained_variance(L, Var)
% Compute cumulative sum of explained variance
exp_var     = ; %Equation 8
cum_var = ; %Equation 9
% Choose p wrt. the Desired Explained Variance
p      = ; %Equation 10
```

Implementation Hint: `Useful function cumsum().`

Test Implementation

To test the `explained_variance.m` function, run the **final** code block in the MATLAB script `test_pca_9d.m`. This should generate both Figure 9, corresponding to a plot for the cumulative explained variance, and Figure 10, corresponding to the projected dataset with the function implemented in Part 1, with $p = 5$ automatically chosen from `explained_variance.m` with $\sigma = 0.9$.

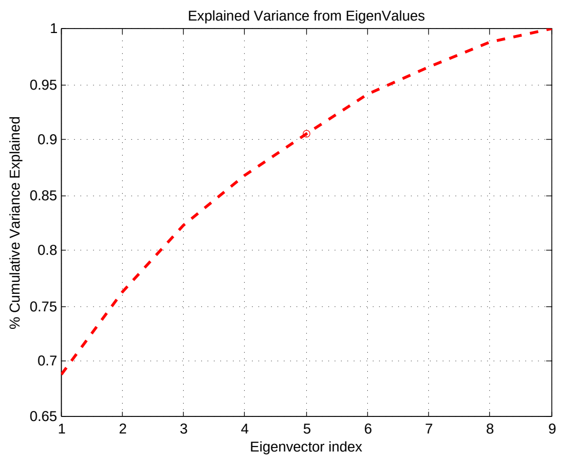

Figure 9: Plot of Cumulative Explained Variance

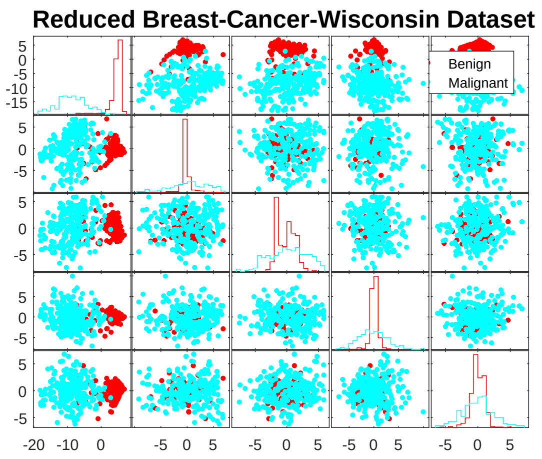

Figure 10: Projected Dataset with p=5

2.2 PCA for Image compression

We can use the ideas behind PCA to perform image compression. Images are a combination of side by side pixels of color values. As an image is not made of random colors and shapes it is reasonable to think that there should be some underlying structure. In other words, if given a pure white pixel in an image, the chances that the surrounding pixels will be white or some variant of white is probably high. Compressing an image is simply removing superfluous pixels. In order not to loose quality we need to find a subspace that requires less pixels to be stored while still being able to perform a reconstruction at the same quality level as the original data. Hence the usage of PCA functionality.

Image compression

An image is a *width* $\times$ *height* $\times$ 3 matrix of float values where the last dimension corresponds to the Red, Green and Blue color channels respectively. To simplify the calculation you will perform the PCA on each channel individually considering that they are independent. You will also perform the PCA over the *width*, i.e you don't need to reshape or transpose the image. In the `compress_image` function, perform the following steps for each channels:

1 Extract the image matrix ($\times height \times$) for the current channel.

2 Compute the PCA over this matrix.

3 Project the PCA on the desired numbers of components.

4 Save the compressed image, the projection and the mean vector for reconstruction.

TASK 7: Implement compress_image function (2pts)

```
function [cimg, ApList, muList] = compress_image(img, p)
% initialize cimg, ApList and muList with the correct dimensions
cimg = ;
ApList = ;
muList = ;
% for all the channels compute the PCA
for i=1:3
    % perform all the steps for compression.
    % save everything in their placeholders
end
```

Compression rate

After PCA, we can estimate the memory necessary to store the compressed image. Complete the `compression_rate` function by calculating the size of the original image, the size of the compressed image and the compression ratio. Be aware that storing only the compressed image is not sufficient so do not forget to also account for all the elements necessary for reconstruction.
TASK 8: Implement compression_rate function (1pts)

```
function [cr, compressedSize] = compression_rate(img,cimg,ApList,muList)
% calculate the bitsize of the original and compressed images
origSize = ;
compressedSize = ;
% calculate the compression ratio
cr = ;
```

Image reconstruction

The final step is the image decoding. This will inverse the PCA to revert the compressed image to the same shape as the original picture. Simply make use of the `reconstruct_pca` function.
TASK 9: Implement reconstruct_image function (1pts)

```
function [rimg] = reconstruct_image(cimg, ApList, muList)
% initialize rimg the correct dimensions
rimg = ;
% for all the channels reconstruct the image
for i=1:3
    % call the reconscrution function and store the reconstructed image
end
```

Test Implementation

Once you have completed all the previous step, you can run the script `test_pca_compression.m`. If everything is correct you should see an output similar to Fig. 11

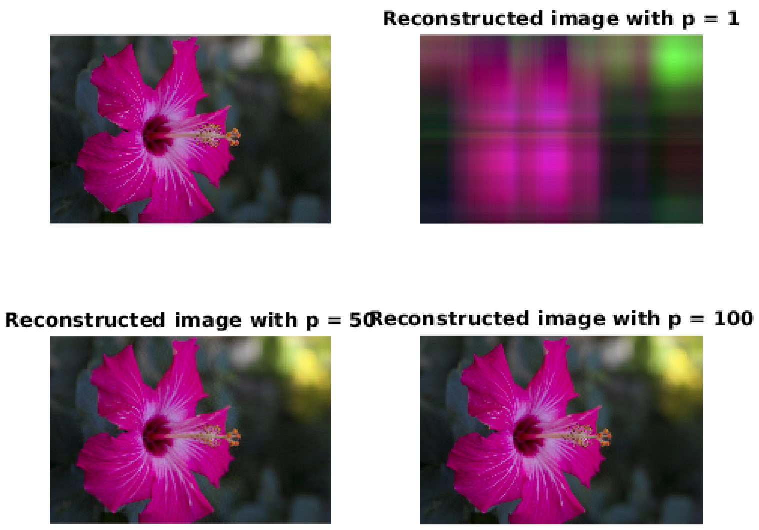
Reconstructed image with p = 1
Reconstructed image with p = 50
Reconstructed image with p = 100

Figure 11: Compression of the image with different values of p

The script should also print the estimate of the storage size for each level of compression:

```
Image compressed by 0.944444%, storage size is 0.55mb
p=1, image compressed by 0.998457%, storage size is 0.02mb
p=50, image compressed by 0.944444%, storage size is 0.55mb
p=100, image compressed by 0.889330%, storage size is 1.09mb
```