| **M24-DS-01 HDDA: Midterm Exam - Practice** | | | Marks obtained ↓ |
|---|---|---|---|
| Date: 17.10.2024, | Total questions: **3** | Total points: **50** | |
| Roll No: | Name: | | Time: 1.5 hrs |

| Question: | 1 | 2 | 3 | Total |
|---|---|---|---|---|
| Points: | 20 | 20 | 10 | 50 |
| Score: | | | | |

**Instructions:**

1. **Create a local colab or jupyter notebook for coding tasks. When you have to answer certain questions, write the text of your answers in the cells with the question ID. (e.g. Question 2.(b)).**
2. **Explain all steps clearly with proper comments**
3. **You cannot have access to lecture notes and labs materials. The elementary pieces of python code are provided.**
4. **The communication between students and the use of external resources (automated such as GPT or not) is automatically sanctioned with a D.**

1. **Q1: Principal component analysis (PCA)** - Detailed calculation of a PCA (based on theory)

   For the data in Table 1, we ask you to perform the following operations "by hand" (you can use a calculator, or program each step, but not use a single "PCA" function that calculates everything at once).

| length [mm] $x_1^{(i)}$ | width [mm] $x_2^{(i)}$ |
|---|---|
| 191 | 155 |
| 195 | 149 |
| 181 | 148 |
| 183 | 153 |
| 176 | 144 |
| 208 | 157 |
| 189 | 150 |
| 197 | 159 |
| 188 | 152 |
| 192 | 150 |
| 179 | 158 |
| 183 | 147 |
| 174 | 150 |
| 190 | 159 |
| 188 | 151 |
| 163 | 137 |
| 195 | 155 |
| 186 | 153 |
| 181 | 145 |
| 175 | 140 |
| 192 | 154 |
| 174 | 143 |
| 176 | 139 |
| 197 | 167 |
| 190 | 163 |

**Table 1: Length and width of a newborn's skull**

Python help :

```
import numpy as np

# Length and width of a newborn's skull
data = [191, 155, 195, 149, 181, 148, 183, 153, 176, 144, 208, 157, 189, 150,
197, 159, 188, 152, 192, 150, 179, 158, 183, 147, 174, 150, 190, 159, 188,
151, 163, 137, 195, 155, 186, 153, 181, 145, 175, 140, 192, 154, 174, 143,
176, 139, 197, 167, 190, 163]
x1 = [data[i] for i in range(0,len(data),2)]
x2 = [data[i] for i in range(1,len(data),2)]
```

[2] (a) Compute the mean vector $\bar{x}$ and center the initial data. Do you think it is required to scale the data ?

[7] (b) Calculate the sample covariance matrix $A$:

- performing operations manually ($cov(X,Y) = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])] = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y]$) where $\mathbb{E}[.]$ denotes the expected value operator.
- by performing the matrix operations $X^T X/m$ with $m$ denoting the number of data samples (i.e., the size of the training set).
- with the *cov* operation in the *numpy* package

If significant differences appear, explain why. If yes, does it have an impact on the computation of the top $k$ principal components of data matrix $X \in \mathbb{R}^{m \times n}$. (If you can't answer this question, move on to the others.)

[1] (c) calculate the eigenvalues and eigenvectors of $A$ using the command *np.linalg.eig(A)*.

[1] (d) compute the percentage of the variance explained by each eigenvalue.

[5] (e) Plot the point cloud $x^{(i)} = (x_1^{(i)}, x_2^{(i)})^T \in \mathbb{R}^2$ with $1 \leq i \leq m$ and represent the axes corresponding to the principal components $(v_1, v_2)$ on the point cloud. To avoid distorting the perception of proximity (or perpendicularity between components), use identical scales on the abscissa and ordinate.
Python help :

- Point cloud from $x$ and $y$:

```
import matplotlib.pyplot as plt

plt.scatter(x, y, s = 20, c = 'red', marker = '+', linewidth = 2)
plt.show()
```

- Draw a straight line $y = ax + b$ with the *abline(a,b)* command.

```
def abline(slope, intercept):
            axes = plt.gca()
            x_vals = np.array(axes.get_xlim())
            y_vals = intercept + slope * x_vals
            plt.plot(x_vals, y_vals, '--')
```

- to set the ratio between axes:

```
ax = plt.gca()
       ax.set_aspect(1)
```

- to draw some vectors $z \in \mathbb{R}^2$, use for instance the quiver function from *matplotlib* :

```
origin = np.array([[0, 0],[0, 0]]) # origin point
plt.quiver(*origin, abscissa of z, ordinate of z, color=['b','g'], scale=5)
```

[3] (f) Calculate the linear regression for $x_2$ from $x_1$ (the best in the least squares sense), i.e. the regression line $x_2 = \theta_1 x_1 + \theta_0$ minimizing the squared $\ell_2$-norm of the prediction errors vector $e$ such that $e_i = x_2^{(i)} - (\theta_1 x_1^{(i)} + \theta_0)$ for $1 \leq i \leq m$.

[1] (g) Is this regression line identical to the first principal axis? Explain.

2. **Q2: SVD and image compression**

Singular value decomposition (SVD) has many applications, for example in image compression. The aim of this exercise is to illustrate this, and to familiarize yourself with some rudiments of image processing in Python. Some pieces of code are provided to help you.

An **image** (in black and white) is in fact a **matrix** (denoted $A$ in the SVD lecture notes) whose values represent the intensity of a pixel.

**Preparation of the data**

2 (a) First load the image (in color: RGB = red, green, blue) and convert it into a black & white image, then into an $A$ matrix with *numpy*, and check visually.

```
# Import useful modules
import matplotlib.pyplot as plt
import numpy as np
from skimage.color import rgb2gray
from skimage import data

images = (
    'astronaut',
    'binary_blobs',
    'brick',
    'colorwheel',
    'camera',
    'cat',
    'checkerboard',
    'clock',
    'coffee',
    'coins',
    'eagle',
    'grass',
    'gravel',
    'horse',
    'logo',
    'page',
    'text',
    'rocket',
)

# We select the "coffee" image
caller = getattr(data, 'coffee')
original = caller()
# Then we convert it in grayscale
grayscale = rgb2gray(original)
# Plot
fig, axes = plt.subplots(1, 2, figsize=(8, 4))
ax = axes.ravel()
ax[0].imshow(original)
ax[0].set_title("Original")
ax[1].imshow(grayscale, cmap=plt.cm.gray)
ax[1].set_title("Grayscale")
fig.tight_layout()
plt.show()
```

```
# We now convert the image into a matrix for processing with numpy
A = np.matrix(grayscale)
```

[1] (b) Perform an SVD on this matrix $A$, to determine the components $U, S, V$ using the *svd* command in numpy's *linalg* library.

[4] (c) Calculate an (approximate!) reconstruction of the image using only the first $k$ rank-1 factors. Visually check that the result corresponds (roughly) to the main features of the image (e.g. the cup, the teaspoon next to the cup of coffee, $\cdots$). Write the formula for $A_k$. (the best rank-k approximation of $A$ with respect to any unitarily invariant norm).

[4] (d) Reconstruct with the first $k = 2, 3, 4, 5, 10, 20, 30$ and 100 rank-1 factors and display the results.

[5] (e) When is a satisfactory reconstruction possible (for which value of $k$)? Support your claim by showing the cumulative singular values plot for the the different values of $k$.
What kind of information bring the rank-1 factors for $k$ in the interval $[20, 30]$ to the approximation ? Display the result.

[4] (f) Estimate the corresponding compression ratio.

[10] 3. **Q3: Hyperspectral Image Segmentation**

Full details can be found in the "Q3_midterm_HDDA" archive available in the shared repository. Open the *Segmentation.ipynb* file and follow the step-by-step instructions.