

# CS 422/622 Project 4

Due 11/29 11:59pm

**Logistics: Logistics:** You must implement everything stated in this project description that is marked with an **implement** tag. **You may only use numpy and matplotlib.pyplot. No late work will be accepted.** I advise that you submit to Webcampus regularly and well before the deadline.

**Deliverables:** Each student should submit a single ZIP file, containing your project code (\*.py files) and your README. Your zip file should be named lastname\_firstname\_project4.zip. For example, a zip file for Sara Smith might look like smith\_sara\_project4.zip. Your code should run without errors on the ECC linux machines. If your code does not run for a particular problem, you will lose 50% on that problem. **If you submit code that plots/prints things, you will be docked points.**

## 1 PCA (50 points)

**File name:** pca.py

**Implement:** You will implement four functions listed here and detailed below.

```
def compute_Z(X, centering=True, scaling=False)
def compute_covariance_matrix(Z)
def find_pcs(COV)
def project_data(Z, PCS, L, k, var)
```

```
def compute_Z(X, centering=True, scaling=False)
```

The above function will take the data matrix  $X$ , and boolean variables `centering` and `scaling`.  $X$  has one sample per row. Remember there are no labels in PCA. If `centering` is True, you will subtract the mean from each feature. If `scaling` is True, you will divide each feature by its standard deviation. This function returns the  $Z$  matrix (numpy array), which is the same size as  $X$ .

```
def compute_covariance_matrix(Z)
```

The above function will take the standardized data matrix  $Z$  and return the covariance matrix  $Z^T Z = \text{COV}$  (a numpy array).

```
def find_pcs(COV)
```

The above function will take the covariance matrix  $\text{COV}$  and return the ordered (largest to smallest) principal components  $\text{PCS}$  (a numpy array where each column is an eigenvector) and corresponding eigenvalues  $L$  (a numpy array). You will want to use `np.linalg.eig` for this.

```
def project_data(Z, PCS, L, k, var)
```

The above function will take the standardized data matrix  $Z$ , the principal components  $\text{PCS}$ , and corresponding eigenvalues  $L$ , as well as a `k` integer value and a `var` floating point value. `k` is the number of principal components you wish to maintain when projecting the data into the new space.  $0 \leq k \leq D$ . If `k` = 0, then we use the cumulative variance to determine the projection dimension. `var` is the desired cumulative variance explained by the projection.  $0 \leq v \leq 1$ . If `v` = 0, then `k` is used instead. Assume they are never both 0 or both > 0. This function will return  $Z_{\text{star}}$ , the projected data.

## 2 Application (50 points)

**File name:** compress.py

You will use grayscale face images for this application. They are located in the `DATA/TRAIN` and `DATA/TEST` directories provided with the project description. You may test on both of them.

**Implement:** You will implement two functions listed here and detailed below.

```
def compress_images(DATA,k)
def load_data(input_dir)
```

You will write an image compression function detailed below.

```
def compress_images(DATA,k)
```

This function will take the flattened image data (**DATA**) as input, along with **k**, the number of principal components to use. This function will use PCA to find the principal components of the face images. It will return the compressed data.  $X_{compressed} = Z^*U^T$ . Where  $Z^*$  is the projected data and  $U^T$  is the transpose of the principal components. This function will then output the compressed images into a directory named **Output**. You should use the **os** package to make sure that the directory is present and to create it if it does not exist. NOTE: Images have values from 0 to 255, so you will want to rescale them before saving them. You will also want to use the **cmap='gray'** option in **pyplot.imshow** to save them as grayscale images.

You will write the following helper functions:

```
def load_data(input_dir)
```

The above function takes the input directory as input, and outputs the **DATA** matrix. **DATA** will have one flattened image per column, so each column represents an image and one row represents the pixel values for every image at a particular location. This function will use **pyplot.imread** to load the images. **Before you return DATA you will want to convert it to floating point.**