HW3: Principal Component Analysis

Due Feb 16 by 2:30pm **Points** 100 **Submitting** a file upload **File Types** zip
Available Feb 9 at 4pm - Feb 16 at 3pm 7 days

This assignment was locked Feb 16 at 3pm.

Assignment Goals

- Explore Principal Component Analysis (PCA) and the related Python packages
- Make pretty pictures :)

Summary

In this project, you'll be implementing a facial analysis program using Principal Component Analysis (PCA), using the skills you learned from the **linear algebra + PCA** lecture. You'll also continue to build your Python skills.

We'll walk you through the process step-by-step (at a high level).

Packages Needed for this Project

In this project, you'll need the following packages:

```
>>> from scipy.linalg import eigh
>>> import numpy as np
>>> import matplotlib.pyplot as plt
```

Dataset

You will be using part of **Yale face dataset (http://vision.ucsd.edu/~leekc/ExtYaleDatabase/ExtYaleB.html)** (processed). You can download the dataset here: **YaleB_32x32.npy (https://canvas.wisc.edu/courses/230450/files/17958123/download?download_frd=1)**

The dataset contains *2414* sample images, each of size *32x32*. We will use *n* to refer to number of images, **n=2414** and *d* to refer to number of features for each sample image, **d=1024** (32x32). We will test your code only using the provided data set. Note, we'll use x_i to refer to the ith sample image which would be a d-dimensional feature vector.

Program Specification

Implement these **six** Python functions to do PCA on our provided dataset, in a file called **pca.py (https://canvas.wisc.edu/courses/230450/files/18030243/download?download_frd=1)** :

1. **load_and_center_dataset(filename) —** load the dataset from a provided .npy file, re-center it around the origin and **return** it as a NumPy array of floats
2. **get_covariance(dataset) —** calculate and **return** the covariance matrix of the dataset as a NumPy matrix (d x d array)
3. **get_eig(S, m) —** perform eigen decomposition on the covariance matrix S and **return** a diagonal matrix (NumPy array) with the largest m eigenvalues on the diagonal, *and* a matrix (NumPy array) with the corresponding eigenvectors as columns
4. **get_eig_perc(S, perc) —** similar to get_eig, but instead of returning the first m, return all eigenvalues and corresponding eigenvectors in similar format as get_eig that explain more than *perc* % of variance
5. **project_image(image, U) —** project each image into your m-dimensional space and **return** the new representation as a d x 1 NumPy array
6. **display_image(orig, proj) —** use matplotlib to display a visual representation of the original image and the projected image side-by-side

Load and Center the Dataset

First, if you haven't, download our sample dataset to the machine you're working on: **YaleB_32x32.npy (https://canvas.wisc.edu/courses/230450/files/17958123/download?download_frd=1)** .

Once you have it, you'll want to use the numpy function `load()` to load the file into Python. (You may need to install NumPy first.)

```
>>> x = np.load(filename)
```

This should give you an *n x d* dataset (*n:* the number of images in the dataset and *d:* the dimensions of each image)

Each row represents an image feature vector. For this particular dataset, we have n = **2414** (no. of images) and d=32 x 32=**1024** (32 by 32 pixels).

Your next step is to center this dataset around the origin. Recall the purpose of this step from lecture---it is a technical condition that makes it easier to perform PCA---but does not lose any information.

To center the dataset is simply to subtract the mean μ_x from each data point x_i (image in our case), i.e. $x_i^{cent} = x_i - \mu_x$ where mean $\mu_x = \frac{1}{n}\sum_{i=1}^{n} x_i$ (n is the no. of images)

You can take advantage of the fact that `x` (as defined above) is a NumPy array and as such, has this convenient behavior:

```
>>> x = np.array([[1,2,5],[3,4,7]])
>>> np.mean(x, axis=0)
=> array([2., 3., 6.])
>>> x - np.mean(x, axis=0)
=> array([[-1., -1., -1.],
          [ 1.,  1.,  1.]])
```

After you've implemented this function, it should work like this:

```
>>> x = load_and_center_dataset('YaleB_32x32.npy')
>>> len(x)
=> 2414
>>> len(x[0])
=> 1024
>>> np.average(x)
=> -8.315174931741023e-17
```

(Its center isn't *exactly* zero, but taking into account precision errors over 2414 arrays of 1024 floats, it's what we call Close Enough.)

Note, From now on, we will use x_i to refer to x_i^{cent}.

Find Covariance Matrix

Recall, from lecture, that one of the interpretations of PCA is that it is the eigendecomposition of the sample covariance matrix. We will rely on this interpretation in this assignment, with all of the information you need below.

The covariance matrix is defined as

$$S \ = \frac{1}{n-1}\sum_{i=1}^{n} \ x_i x_i^T$$

Note that x_i is one of the n images in the dataset (centered) and it is considered as a column vector (size **d x 1**) in this formula.

If you consider it as a row vector (size **1 x d**), then the formula would like below.

$$S \ = \frac{1}{n-1}\sum_{i=1}^{n} \ x_i^T x_i$$

To calculate this, you'll need a couple of tools from NumPy again:

```
>>> x = np.array([[1,2,5],[3,4,7]])
>>> np.transpose(x)
=> array([[1, 3],
          [2, 4],
          [5, 7]])
>>> np.dot(x, np.transpose(x))
=> array([[30, 46],
          [46, 74]])
>>> np.dot(np.transpose(x), x)
=> array([[10, 14, 26],
          [14, 20, 38],
          [26, 38, 74]])
```

The result of this function for our sample dataset should be a **d x d** (1024 x 1024) matrix.

Get m Largest Eigenvalues/Eigenvectors

Again, recall from lecture that eigenvalues and eigenvectors are useful objects that characterize matrices. Better yet, PCA can be performed by doing an eigendecomposition and taking the eigenvectors corresponding to the largest eigenvalues. This replaces the recursive deflation step we discussed in class.

You'll never have to do eigen decomposition by hand again (https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.eigh.html) ! Use `scipy.linalg.eigh()` to help, particularly the optional `eigvals` argument.

We want the *largest m eigenvalues* of S.

Return the eigenvalues as a diagonal matrix, in descending order, and the corresponding eigenvectors as columns in a matrix.

To return more than one thing from a function in Python:

```
def multi_return():
    return "a string", 5
mystring, myint = multi_return()
```

Make sure to return the diagonal matrix of eigenvalues FIRST, then the eigenvectors in corresponding columns. You may have to rearrange the output of eigh() to get the eigenvalues in decreasing order and *make sure to keep the eigenvectors in the corresponding columns* after that rearrangement.

```
>>> Lambda, U = get_eig(S, 2)
>>> print(Lambda)
[[1369142.41612494       0.        ]
 [      0.         1341168.50476773]]
>>> print(U)
[[-0.01304065 -0.0432441 ]
[-0.01177219 -0.04342345]
[-0.00905278 -0.04095089]
...
[ 0.00148631 0.03622013]
[ 0.00205216 0.0348093 ]
[ 0.00305951 0.03330786]]
```

Get all Eigenvalues/Eigenvectors that Explain More than Certain % of Variance

We want *all* the *eigenvalues* that explain more than a certain percentage of variance.

Let λ_i be an eigenvalue of the covariance matrix **S**. Then the percentage of variance explained is calculated as:

$$\frac{\lambda_i}{\sum_{i=1}^{n} \lambda_i}$$

Return the eigenvalues as a diagonal matrix, in descending order, and the corresponding eigenvectors as columns in a matrix.

Make sure to return the diagonal matrix of eigenvalues FIRST, then the eigenvectors in corresponding columns. You may have to rearrange the output of eigh() to get the eigenvalues in decreasing order and *make sure to keep the eigenvectors in the corresponding columns* after that rearrangement.

```
>>> Lambda, U = get_eig_perc(S, 0.07)
>>> print(Lambda)
[[1369142.41612494       0.        ]
 [      0.         1341168.50476773]]
>>> print(U)
[[-0.01304065 -0.0432441 ]
[-0.01177219 -0.04342345]
[-0.00905278 -0.04095089]
...
[ 0.00148631 0.03622013]
[ 0.00205216 0.0348093 ]
[ 0.00305951 0.03330786]]
```

Project the Images

Given one of the images from your dataset and the results of your `get_eig()` function, create and return the *projection* of that image.

For any image x_i, we project it into the *m* dimensional space as

$x_i^{pro} = \sum_{j=1}^{m} \alpha_{ij} u_j$ where $\alpha_{ij} = u_j^T x_i$, $\alpha_i \in \mathbb{R}^m$ and $x_i^{pro} \in \mathbb{R}^d$.

u_j is an eigenvector column (size **d x 1**) of U from the previous function.

Find the alphas for your image, then use them together with the eigenvectors to create your projection.

```
>>> projection = project_image(x[0], U)
>>> print(projection)
[6.84122225 4.83901287 1.41736694 ... 8.75796534 7.45916035 5.4548656 ]
```

Visualize

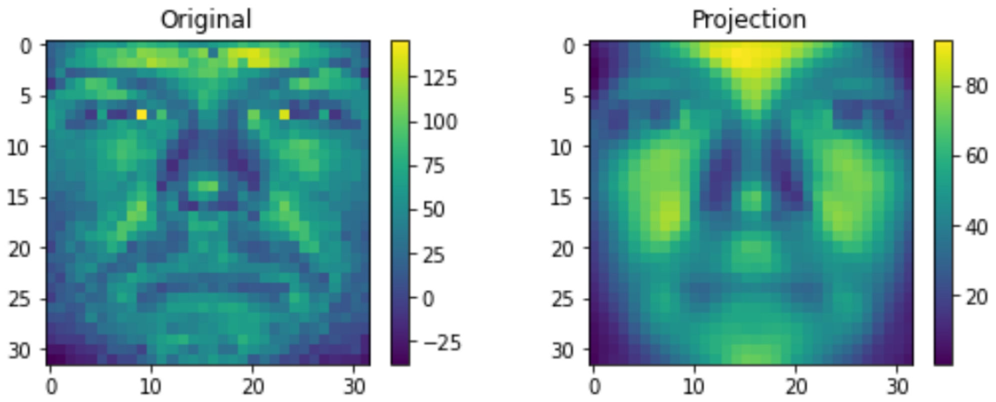
We'll be using **matplotlib's imshow (https://matplotlib.org/3.1.3/api/_as_gen/matplotlib.pyplot.imshow.html)**. First, make sure you have the **matplotlib library (https://matplotlib.org/3.1.1/users/installing.html)**, for creating figures.

Follow these steps to visualize your images:

1. Reshape the images to be 32x32 (you should have calculated them as 1d vectors of 1024 numbers).

2. Create a figure with one row of two **subplots** **(https://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplots.html)** .
3. The first subplot (on the left) should be **titled** **(https://matplotlib.org/3.1.3/gallery/subplots_axes_and_figures/figure_title.html)** "Original", and the second (on the right) should be titled "Projection".
4. Use `imshow()` with optional argument `aspect='equal'` to render the original image in the first subplot and the projection in the second subplot.
5. Use the return value of `imshow()` to create a **colorbar** **(https://matplotlib.org/3.1.0/gallery/color/colorbar_basics.html)** for each image.
6. **Render** **(https://matplotlib.org/api/_as_gen/matplotlib.pyplot.show.html)** your plots!

```
>>> x = load_and_center_dataset('YaleB_32x32.npy')
>>> S = get_covariance(x)
>>> Lambda, U = get_eig(S, 2)
>>> projection = project_image(x[0], U)
>>> display_image(x[0], projection)
```



Submission Notes

Please submit your files in a zip file named **hw3_<netid>.zip**, where you replace <netid> with your netID (your wisc.edu login). Inside your zip file, there should be **only** one file named: **pca.py**. Do not submit a Jupyter notebook .ipynb file.

Be sure to **remove all debugging output** before submission; your functions should run silently (except for the image rendering window). Failure to remove debugging output will be **penalized (10pts)**.

This assignment is due on **02/16/2021 2:30pm**. It is preferred to first submit a version well before the deadline and check the content/format of the submission to make sure it's the right version. Then, later update the submission until the deadline if needed.

PCA Rubric

Criteria	Ratings		Pts
load_and_center_dataset() returns correct output	**20 pts** **Full Marks**	**0 pts** **No Marks**	20 pts
get_covariance() returns correct result for the provided dataset	**15 pts** **Full Marks**	**0 pts** **No Marks**	15 pts
get_eig() & get_eig_perc() return correct eigenvalue and eigenvector matrix	**25 pts** **Full Marks**	**0 pts** **No Marks**	25 pts
project_image() correctly projects image vector onto eigenvectors	**15 pts** **Full Marks**	**0 pts** **No Marks**	15 pts
display_image() creates correctly formatted plot (titles/subplots/colorbars appear as specified)	**10 pts** **Full Marks**	**0 pts** **No Marks**	10 pts
display_image() displays images correctly for given input	**15 pts** **Full Marks**	**0 pts** **No Marks**	15 pts
			Total Points: 100