

# HW5: Principal Component Analysis

**Due** Oct 27 by 11:05am    **Points** 100    **Submitting** a file upload  
**Available** until Oct 27 at 11:05am

This assignment was locked Oct 27 at 11:05am.

## Assignment Goals

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

## Summary

In this project, you'll be implementing a handwriting analysis program using Principal Component Analysis (PCA).

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

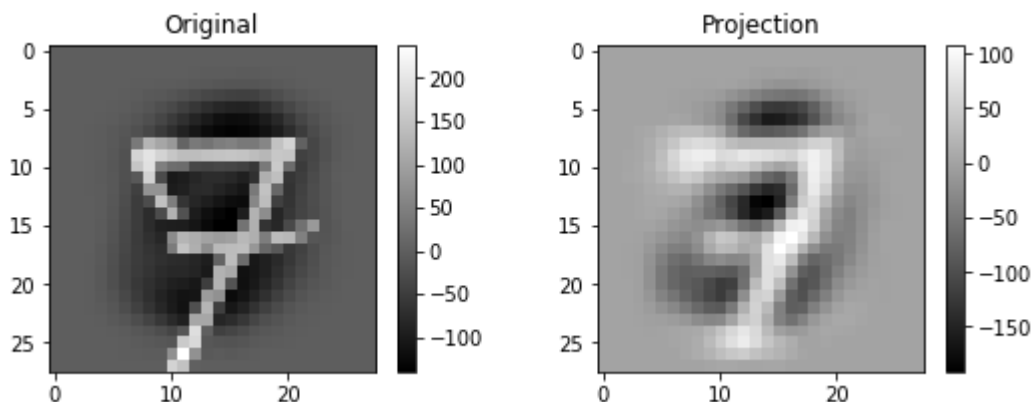
## Packages Needed for this Project

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

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

## Dataset

You'll be using part of the MNIST dataset [mnist.npy](#).



The dataset contains 2000 samples images, each of size  $28 \times 28$ . We will use  $n$  to refer to number of samples where  $n=2000$ . And  $d$  to refer to number of features for each sample which is  $d=28 \times 28=784$ . We will only test your code using this data set. Here we'll use  $x_i$  to refer to the  $i$ th sample which would be a  $d$  dimension column vector.

## Program Specification

Implement these **six (6)** Python functions to do PCA on our provided dataset, in a file called **pca.py**:

1. **load\_and\_center\_dataset(filename)** — load the dataset from a provided .npz 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 ( $d \times d$  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 eigenvectors that explains 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  $d \times 1$  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

**NOTE: your orig, proj, image and return value of project\_image should have np.shape as (784,)**

## Load and Center the Dataset

First, if you haven't, download our sample dataset to the machine you're working on: [mnist.npz](#).

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)
```

Then, you should reshape the data to  $n \times d$  ( $n$ , the number of images we're analyzing, and  $d$ , the dimension of those images). Here we have  $d=28 \times 28=784$  (28 by 28 pixels).

```
>>> x = np.reshape(x, (2000, 784))
```

This should give you an  $n \times d$  dataset, where each **row** is an image feature vector of 784 pixels. Note this configuration for future calculations.

Your next step is to re-center this dataset around the origin. Re-centering is simply to subtract the dataset mean  $\mu_{x^{orig}}$  from each data point  $x_i^{orig}$ . This is defined as  $x_i^{cent} = x_i^{orig} - \mu_{x^{orig}}$  where  $\mu_{x^{orig}} = \frac{1}{n} \sum_i x_i^{orig}$

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

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('mnist.npy')
>>> len(x)
=> 2000
>>> len(x[0])
=> 784
>>> np.average(x)
=> -2.5057507089662307e-16
```

Its center isn't *exactly* zero, but taking into account precision errors over 2000 arrays of 784 floats, it's what we call Close Enough. (`np.average(x)` have absolute value below  $10^{-10}$ )

## Find Covariance Matrix

As given in the notes, the covariance matrix is defined as (if you consider ith sample  $x_i$  as a column vector)

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

(if you consider ith sample  $x_{row\ i}$  as a row vector, then it is)

$$S = \frac{1}{n-1} \sum_i x_{row\ i}^T x_{row\ i}$$

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  $d \times d$  (784x784) matrix.

## Get m Largest Eigenvalues/Eigenvectors

[You'll never have to do eigen decomposition by hand again](https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.eigh.html)

(<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)
[[350880.76329673      0.
  [      0.      245632.27295307]]
>>> np.sum(U)
12.594115166027581
```

## Get all Eigenvectors that Explain More than Certain % of Variance

We want all the *eigenvalues* that explain more than a certain percentage of variance. Let  $u_i$  and  $\lambda_i$  be an eigenvector and corresponding eigenvalue.

Then the percentage of variance explained is:

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)
[[ 350880.76329673      0.      ]
 [      0.      245632.27295307]]
>>> np.sum(U)
12.594115166027581
```

## 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  $\mathbf{x}_i$ , we project it into the  $m$  dimensional space span eigenvectors by:

where  $\alpha_i \in \mathbb{R}^m$  and  $\mathbf{U} \in \mathbb{R}^{n \times m}$ .

## Visualize

We'll be using [matplotlib's imshow](https://matplotlib.org/3.1.3/api/_as_gen/matplotlib.pyplot.imshow.html)

([https://matplotlib.org/3.1.3/api/\\_as\\_gen/matplotlib.pyplot.imshow.html](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) (<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 28x28 (you should have calculated them as 1d vectors of 784 numbers).
2. Create a figure with one row of two [subplots](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplots.html) ([https://matplotlib.org/api/\\_as\\_gen/matplotlib.pyplot.subplots.html](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](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'` and `cmap='gray'` 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](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](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.show.html)) your plots!

NOTE: to plot image on CSL machines, you will need to save the plot to a file (cause it doesn't have graphic interface)

1. add the following lines to the beginning of your code.

```
>>> import matplotlib
```

```
>>> matplotlib.use('Agg')
```

2. instead of using `plt.show()` use `plt.savefig("filename.png")`.

(You should use `plt.show()` in your submission.)

## Submission Notes

Please submit your files in a zip file named **hw5\_<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)**.

**If a regrading request isn't justifiable (the initial grade is correct and clear, subject to instructors' judgment)**, the request for regrading will be **penalized (10 pts)**.

**This assignment due at 10/27/2020 11:00am. Submitting right at 11:00am will result in a late submission. It is preferred to first submit a version well before the deadline (at least one hour before) 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.**

# Changelog

See below for updates to the assignment since the initial release

Added explanation for  $n$  and  $d$  in dataset section.

Updated to correct the diagonal flipped sample image.

Elaborate that  $\mathbf{x}_i$  is column vector.

Elaborate on the definition of covariance matrix as row vectors.

Add clarification that students will only be tested on mnist.npy

Added clarification on `np.shape()` of vectors

Added guide about how to test plot on CSL machines

Changed the CSL guide from jpg to png

## Rubric

Criteria	Ratings		Pts
load_and_center_dataset	<b>20.0 pts</b> <b>Full Marks</b>	<b>0.0 pts</b> <b>No Marks</b>	20.0 pts
get_covariance	<b>15.0 pts</b> <b>Full Marks</b>	<b>0.0 pts</b> <b>No Marks</b>	15.0 pts
get_eig & get_eig_perc	<b>25.0 pts</b> <b>Full Marks</b>	<b>0.0 pts</b> <b>No Marks</b>	25.0 pts
project_image	<b>15.0 pts</b> <b>Full Marks</b>	<b>0.0 pts</b> <b>No Marks</b>	15.0 pts
display_image	<b>15.0 pts</b> <b>Full Marks</b>	<b>0.0 pts</b> <b>No Marks</b>	15.0 pts
image correctly formated	<b>10.0 pts</b> <b>Full Marks</b>	<b>0.0 pts</b> <b>No Marks</b>	10.0 pts
			Total Points: 100.0