ECE 443 (Fall 2022) – Programming Exercise #3 (75 points)

Last updated: November 1, 2022

Rationale and learning expectations: There are so many facets of *Principal Component Analysis* (PCA) that they cannot be covered in one exercise alone. While Exercise #2 focused on the basics of PCA, this exercise is meant to help us understand additional aspects of PCA through processing of different types of synthetic and real-world data. In addition, this exercise introduces students to the basic set of APIs that form the core of the `scikit learn` library for machine learning in Python. At the conclusion of this activity, students attempting this exercise are expected to understand the PCA problem at a deeper level, when compared to Exercise #2, and are also expected to understand the basic workings of the `scikit learn` library.

General Instruction: All parts of this exercise must be done within a Notebook, with text answers (and other discussion) provided in text cells and commented code provided in code cells. Please refer to the solution template for Exercise #3 as a template for your own solution. In particular:

- Replace any text in the text cells enclosed within square brackets (e.g., `[Your answer to 1.1a goes in this cell]`) with your own text using Markdown and/or LaTeX .
- Replace any text in the code cells enclosed within pairs of three hash symbols (e.g., `### Your code for 1.1a goes in this cell ###`) with your own code.
- Unless expressly permitted in the template, **do not** edit parts of the template that are not within square brackets / three hash symbols and **do not** change the cell structure of the template.
- Make sure the submitted notebook is fully executed (i.e., submit the notebook only after a full run of all cells).

Restrictions: You are free to use `numpy`, `pandas`, `scipy.stats`, `matplotlib`, `mpl_toolkits.mplot3d`, `seaborn`, and `IPython.display` packages within your code. Unless explicitly permitted by the instructor, you are not allowed to use any other libraries, packages, or modules within your submission.

Notebook Preamble: I suggest importing the different libraries / packages / modules in the preamble as follows (but you are allowed to use any other names of your liking):

```
import numpy as np
import pandas as pd
from scipy import stats as sps
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from IPython.display import display, Latex
```

1 Non-centered Data and Principal Component Analysis (PCA)

In this part of the exercise, we will work with a three-dimensional synthetic dataset that, unlike the synthetic dataset of Exercise #2, does not originally pass through the origin.

1.1. (1 point) Create a matrix $\mathbf{A} \in \mathbb{R}^{3\times 2}$ whose individual entries are drawn from a Gaussian distribution with mean 0 and variance 1 in an independent and identically distributed (iid) fashion. Also, create a vector $\mathbf{c} \in \mathbb{R}^3$ whose individual entries are iid and drawn from a Gaussian distribution with mean 0 and variance 3. Once generated, both $\mathbf{A}$ and $\mathbf{c}$ should not be changed for the rest of the problems in this section.

1.2. (2 points) Generate a synthetic dataset with 250 data samples $\{\mathbf{x}_i\}_{i=1}^{250}$ as follows. Each data sample $\mathbf{x}_i \in \mathbb{R}^3$ in the dataset is generated as $\mathbf{x}_i = \mathbf{A}\mathbf{b}_i + \mathbf{c}$, where $\mathbf{b}_i \in \mathbb{R}^2$ is a random vector whose entries are iid Gaussian with mean 0 and variance 1. Note that we will have a different $\mathbf{b}_i$ for each data sample $\mathbf{x}_i$ (i.e., unlike $\mathbf{A}$ and $\mathbf{c}$, it is **not** fixed for each data sample). Store the data samples into a *data matrix* $\mathbf{X} \in \mathbb{R}^{250\times 3}$, such that each data sample is a row in this data matrix.

1.3. (1 point) Except for the addition of $\mathbf{c}$ to each data sample, this dataset is identical to the synthetic dataset of Section 2 in Exercise #2 in terms of the generation mechanism. Addition of $\mathbf{c}$, however, shifts our data from the origin (i.e., makes it *non-centered*), which increases its rank. Verify this by printing the rank of $\mathbf{X}$.

1.4. (3 points) While our random data generation mechanism guarantees that $\mathbf{X}$ will have rank 3 (i.e., the data samples no longer pass through the origin), there are choices of $\mathbf{c} \neq \mathbf{0}$ that guarantee that the data remains centered. List at least two such choices and justify your answer.

1.5. Verify the importance of centering the data as an essential preprocessing step for PCA by carrying out the following steps:

(a) (3 points) Compute the top-two principal component directions $\mathbf{U} = \begin{bmatrix}\mathbf{u}_1 & \mathbf{u}_2\end{bmatrix}$ of the dataset *without* centering the data.

(b) (2 points) Reconstruct (approximate) the original data samples $\mathbf{x}_i$ from the PCA loading vectors by computing $\widehat{\mathbf{x}}_i = \mathbf{U}\mathbf{U}^T\mathbf{x}_i$ and storing them as rows in a *reconstructed* data matrix $\widehat{\mathbf{X}}$.

(c) (2 points) Compute the representation error (*aka*, PCA error) $\|\mathbf{X} - \widehat{\mathbf{X}}\|_F^2$ and show that this error is nowhere close to being zero, despite the fact that $\mathbf{X}$ has only two major directions of variation that are given by the columns of $\mathbf{A}$.

1.6. (4 points) Repeat the steps described in Problem 1.5 by first centering the data matrix $\mathbf{X}$ using the empirical mean vector $\mathbf{m}$ and then showing that the approximated data samples $\widehat{\mathbf{x}}_i = \mathbf{U}\mathbf{U}^T(\mathbf{x}_i - \mathbf{m}) + \mathbf{m}$ once again lead to (almost) zero representation error *(Note: You are **not** allowed to use the original vector* $\mathbf{c}$ *in your calculations)*.

2 Preprocessing (Centering) and PCA Using `scikit learn`

`scikit learn` is one of the most popular machine learning library/package for Python and, at its core, it operates using a small set of APIs that are explained at `https://scikit-learn.org/stable/developers/develop.html#api-overview`. We will familiarize ourselves with some of these APIs by focusing on centering and PCA of the dataset of Section 1 using `scikit learn`. In order to proceed with this section, it is recommended that you import various modules within `scikit learn` as follows (but you are allowed to use any other names of your liking):

```
from sklearn import preprocessing as sklpp
from sklearn import decomposition as skldecomp
```

2.1. (2 points) Center the dataset generated in Section 1 by using the following commands (you might want to use the variable names that suit your style and also do not result in writing over of other variables):

```
# create an instance of the StandardScaler() object
mean_datascaler = sklpp.StandardScaler(with_mean=True, with_std=False)
# use the fit_transform API to simultaneously compute mean and center data
skl_centered_X = mean_datascaler.fit_transform(X)
```

2.2. (2 points) Verify by printing that the mean vector $\mathbf{m}$ computed by you in Problem 1.6 and the mean vector computed by `StandardScaler()` in Problem 2.1 are the same vectors. *Hint:* `mean_datascaler.mean_` will return the mean vector calculated by `StandarScaler()` as part of the `fit_transform()` operation.

2.3. (2 points) Let us denote the centered data matrix returned by `StandardScaler()` in Problem 2.1 as $\overline{\mathbf{X}}_{\text{skl}}$ and let us denote the centered data matrix computed by you in Problem 1.6 as $\overline{\mathbf{X}}$. Verify that the two computations yield the same result by computing and printing $\|\overline{\mathbf{X}} - \overline{\mathbf{X}}_{\text{skl}}\|_F^2$.

 waheed.bajwa@rutgers.edu

2.4. (2 points) Compute the PCA directions (*principal axes* / *loading vectors*) and the PCA features of the centered dataset by using the following commands (you might want to use the variable names that suit your style and also do not result in writing over of other variables):

```
# create an instance of the PCA() object
data_pca = skldecomp.PCA(n_components=2, svd_solver='full')
# use fit_transform API to simultaneously compute PCA features & directions
skl_features = data_pca.fit_transform(skl_centered_X)
```

2.5. (2 points) Verify by printing that the two-dimensional PCA subspace computed by you in Problem 1.6 and the one computed by `PCA()` in Problem 2.4 are the same ones. *Hint:* `data_pca.components_` will return the subspace computed by `PCA()` as part of the `fit_transform()` operation.

2.6. (2 points) Compute projections of your data onto the two-dimensional subspace spanned by the top-two principal components returned by `PCA()` and *uncenter* these projections by adding back the mean vector stored within the `StandardScaler()` object created in Problem 2.1. *Hint:* The projection operation can be carried out using the `inverse_transform()` method of the `PCA()` object created in Problem 2.4.

2.7. (2 points) Let us denote the (*non-centered*) projected data matrix obtained using `PCA()` in Problem 2.6 as $\widehat{\mathbf{X}}_{\text{skl}}$ and let us denote the (*non-centered*) projected data matrix computed by you in Problem 1.6 as $\widehat{\mathbf{X}}$. Verify that the two computations yield similar results by computing and printing $\|\widehat{\mathbf{X}} - \widehat{\mathbf{X}}_{\text{skl}}\|_F^2$.

3 PCA on a Real-world Dataset

We now turn our attention to a real-world imaging dataset and explore the use of PCA on such datasets. The particular dataset we focus on in here corresponds to images of handwritten digit '0'; you are allowed to resort to the `StandardScaler()` and `PCA()` objects of `scikit learn` to complete this section. In order to load this data into your notebook, you should use the following code snippet:

```
from sklearn.datasets import load_digits
X, _ = load_digits(1, return_X_y=True)
```

Once you run this code, you will have different 8×8 images of digit 0 stored in the `numpy` array `X` as *flattened* vectors in its rows.

3.1. (2 points) Referring to the notational convention used in the course, what is n (number of samples) and p (dimensionality of each data sample) for this dataset? Also, display any one of the images in the dataset as an 8×8 grayscale image using `matplotlib.pyplot`.

3.2. (3 points) Get the data ready for PCA by mean centering it; display the mean vector as an 8×8 grayscale image using `matplotlib.pyplot`.

3.3. (1 point) Let us denote the *centered* imaging dataset as $\bar{\mathbf{X}}$. Compute the singular value decomposition of $\bar{\mathbf{X}}$ using `numpy.linalg.svd()`.

3.4. (1 point) Plot the singular values of $\bar{\mathbf{X}}$ on a semi-log scale (i.e., the vertical axis should be $\log$ scale).

3.5. (3 points) Using only the singular values computed in Problem 3.4, find (and print) the *smallest* integer k that satisfies $\frac{\sum_{i=1}^{k} \sigma_i^2}{\sum_{i=1}^{\min n,p} \sigma_i^2} \geq 0.90$. In words, this k ensures that the top-k principal components of the dataset capture at least 90% of the energy in the dataset.

3.6. (2 points) The PCA implementation within `scikit learn` also provides a means to compute the k that was computed by hand in Problem 3.5. In order to do this, the `PCA()` object should be initialized as follows:

```
sklearn.decomposition.PCA(n_components=0.90, svd_solver='full')
```


Verify that the solution of Problem 3.5 matches the k returned by `scikit learn`. *Hint:* The `PCA()` object has an attribute `n_components_` that effectively is k once either `fit()` or `fit_transform()` methods are executed.

3.7. (2 points) Display the first three principal component axes of the centered data $\bar{\mathbf{X}}$ as three 8×8 grayscale images using `matplotlib.pyplot`.

3.8. (2 points) Project each mean centered image onto the top three principal component axes of $\bar{\mathbf{X}}$ to obtain a three-dimensional *feature vector* corresponding to each image.

3.9. (2 points) Using `mpl_toolkits.mplot3d`, display a 3D scatterplot of the n three-dimensional feature vectors obtained in Problem 3.8.

3.10. (2 points) Reconstruct (approximate) the original images from the feature vectors (including accounting for the mean centering).

3.11. (3 points) Compute the average representation error (average PCA error) between the reconstructed images and the original images, and express the relationship between this average representation error and the singular values of the mean centered $\bar{\mathbf{X}}$.

3.12. (1 point) Display the reconstructed version of the image that you selected in Problem 3.1 as an 8×8 grayscale image using `matplotlib.pyplot`.

4 PCA as a Denoising Tool

PCA is like the swiss army knife of machine learning. It can be viewed as a feature learning technique, as a dimensionality reduction method (remember, not all feature learning methods reduce dimensionality), and as a decorrelating transform. In addition, PCA can be viewed as a *denoising* procedure, which can be used to reduce noise in many real-world datasets. We will focus on this aspect of PCA in this section; you are allowed to resort to the `PCA()` object of `scikit learn` to complete this section. In order to demonstrate the denoising power of PCA, we will be creating a brand-new synthetic dataset in this section.

4.1. (1 point) Create a vector $\mathbf{a} \in \mathbb{R}^3$ whose individual entries are drawn from a Gaussian distribution with mean 0 and variance 1 in an iid fashion. Once generated, this vector should not be changed for the rest of this section.

4.2. (3 points) Generate a synthetic dataset with 100 data samples $\{\mathbf{x}_i\}_{i=1}^{100}$ as follows. Each data sample $\mathbf{x}_i \in \mathbb{R}^3$ in the dataset is generated as $\mathbf{x}_i = b_i\mathbf{a}$, where $b_i \in \mathbb{R}$ is a Gaussian random variable with mean 0 and variance 4. Note that we will have a different b_i for each data sample $\mathbf{x}_i$. Store the *noiseless* data samples into a *data matrix* $\mathbf{X} \in \mathbb{R}^{100\times 3}$, such that each data sample is a row in this data matrix.

4.3. (2 points) Generate a *noisy* version of the data matrix $\mathbf{X}$ by adding *white Gaussian noise* of variance 0.05 to the noiseless data samples as follows. Each noisy data sample $\mathbf{y}_i \in \mathbb{R}^3$ in the dataset is generated as $\mathbf{y}_i = \mathbf{x}_i + \mathbf{n}_i$, where the noise vector $\mathbf{n}_i$ has its individual entries drawn from a Gaussian distribution with mean 0 and variance 0.05 in an iid fashion. Store the noisy data samples into another data matrix $\mathbf{Y} \in \mathbb{R}^{100\times 3}$, such that each noisy data sample $\mathbf{y}_i$ is a row in this data matrix.

4.4. (2 points) Using `mpl_toolkits.mplot3d`, display two 3D scatterplots in two different colors on the same plot, where one scatterplot corresponds to the noiseless data samples $\mathbf{x}_i$ and the other scatterplot corresponds to the noisy data samples $\mathbf{y}_i$.

4.5. (4 points) *Denoise* the noisy dataset $\mathbf{Y}$ by projecting the noisy data samples $\mathbf{y}_i$ onto the top principal component axis of the noisy data matrix $\mathbf{Y}$. Store the denoised data samples into another data matrix $\widehat{\mathbf{X}} \in \mathbb{R}^{100\times 3}$, such that each denoised data sample $\widehat{\mathbf{x}}_i \in \mathbb{R}^3$ is a row in this data matrix.

4.6. (2 points) Using `mpl_toolkits.mplot3d`, display two 3D scatterplots in two different colors on the same plot, where one scatterplot corresponds to the noiseless data samples $\mathbf{x}_i$ and the other scatterplot corresponds to the denoised data samples $\widehat{\mathbf{x}}_i$.

waheed.bajwa@rutgers.edu

4.7. (3 points) Compute and display the *average error per noisy sample* as follows: $\frac{\|\mathbf{X}-\mathbf{Y}\|_F^2}{100}$. Similarly, compute and display the *average error per denoised sample* as follows: $\frac{\|\mathbf{X}-\widehat{\mathbf{X}}\|_F^2}{100}$.

4.8. (2 points) Describe in your own words as to what lessons can be drawn from this section of the exercise.

