QH_HW1

April 22, 2021

1 HW1

[5]:
```
import numpy as np
import multiprocessing as mp
from matplotlib import pyplot as plt
import cv2
import glob
```

1.1 Table of Contents

1.2 1. Linear algebra basics

Answer true or false for each of the question below and give justification.

(a) A rectangular matrix of size n × m is a linear transformation.
True; lecture 2 states a matrix $A \in \mathbb{R}^{n \times m}$ performs a linear transformation.

$$A : \mathbb{R}^n \to \mathbb{R}^m$$

$$x \to Ax$$

$$\alpha x + \beta y \to A(\alpha x + \beta y) = \alpha Ax + \beta Ay \text{ (Linear Transform)}$$

(b) Only square matrices have Eigenvalue decompositions.
True; lecture 3 clearly states eigen decomopsition is only defined for square matrices.

(c) `Power Method` can be used to find only eigenvectors (and not singular vectors).
False; left singular vectors can be computed as the eigenvectors of $A^T A$

(d) Left (or right) singular vectors are orthogonal to each other.
True; the singular vectors form the orthonormal basis for the four fundamental subspaces.

(e) If A and B are any 2×2 matrices, then $AB = BA$.
False; we can prove this by counterexample.
Let $

$$A = \begin{bmatrix} 3 & 1 \\ 2 & 1 \end{bmatrix} B = \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix}$$

$

$

$$AB = \begin{bmatrix} 3 & 1 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 1 \\ 4 & 1 \end{bmatrix}$$

$

$

$$BA = \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 3 & 1 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 1 \\ 8 & 3 \end{bmatrix}$$

$

[34]:
```
A = np.array([
    [3, 1],
    [2, 1]
])
B = np.array([
    [1, 0],
    [2, 1]
])
print(A @ B)
print(B @ A)
```

```
[[5 1]
 [4 1]]
[[3 1]
 [8 3]]
```

1.3 2. Python practice via statistical concepts

This questions tests your nu- merical computing skills in python by implementing some basic statistics concepts. You are encouraged to use the Alan server and parallel processing techniques for this question. Let X be a random variable that takes values $+1$, -1 with equal probability. That is:

$$P(X = +1) = P(X = -1) = 1/2.$$

Generate N = 10,000 datasets, each of which has n data points. For this simulation, we consider n = {10, 100, 1000, 10000}. (Hint: Write a function that samples from the uniform distribution between 0 and 1. If the result is less than 0.5, set it to -1. Otherwise, set it to 1). Let $X_n^{(i)}$ be the sample average of ith dataset, = E(X) = 0 and ^{2} = Var(X) = 1. n (Hint: Once you compute the sample averages, you will not need the individual data points from each dataset. Therefore, to save memory, you need only store the $X_n^{(i)}$ rather than all the data points. It is highly recommended that you do this to avoid freezing or crashing your computer). Plot and intepret the following:

1.3.1 Set up functions

[7]:
```
def sample_avg(n):
    x = np.random.uniform(0,1,n)
    x = np.where(x < .5, -1, 1)
    sample_avg = np.mean(x)
    return sample_avg

def gen_n(n):
    x = np.full(
        shape=10000,
        fill_value=n,
        dtype=np.int
    )
    return x
```

[8]:
```
n_values = [10, 100, 1000, 10000]
n_values
```

[8]:
```
[10, 100, 1000, 10000]
```

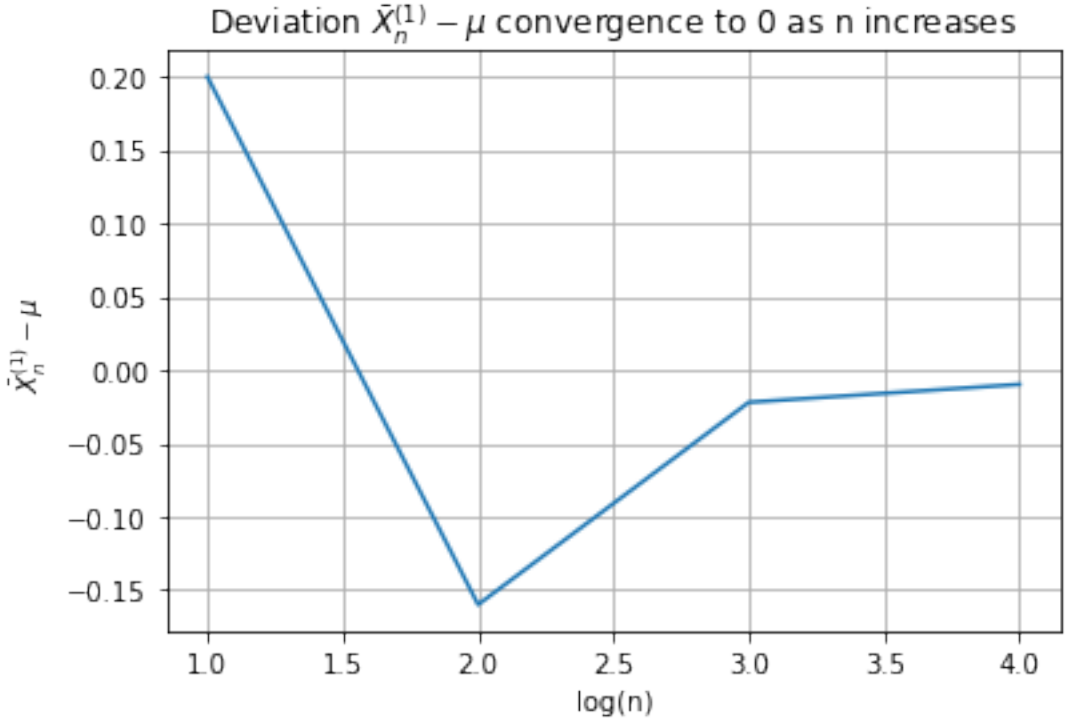
[9]:
```
pool = mp.Pool(processes=4)
results = [pool.map(sample_avg, gen_n(n)) for n in n_values]
results = np.array(results)
m, n = np.shape(results)
np.shape(results)
```

[9]:
```
(4, 10000)
```

1.3.2 (a) Plot $log_{10}(n)$ v.s. $\bar{X}_n^{(1)} - \mu$

[24]:
```
log_n = [np.log10(n) for n in n_values]
log_n

plot_list = [results[n,1] for n in range(4)]
plot_list
```

```
fig, ax = plt.subplots(1, 1)
ax.plot(log_n, plot_list)
ax.set(xlabel = "log(n)",
       ylabel = r"$\bar{X}^{(1)}_n - \mu$ ",
       title = r"Deviation $\bar{X}^{(1)}_n - \mu$ convergence to 0 as n␣
 ↪increases"
      )
ax.grid()
plt.show()
```



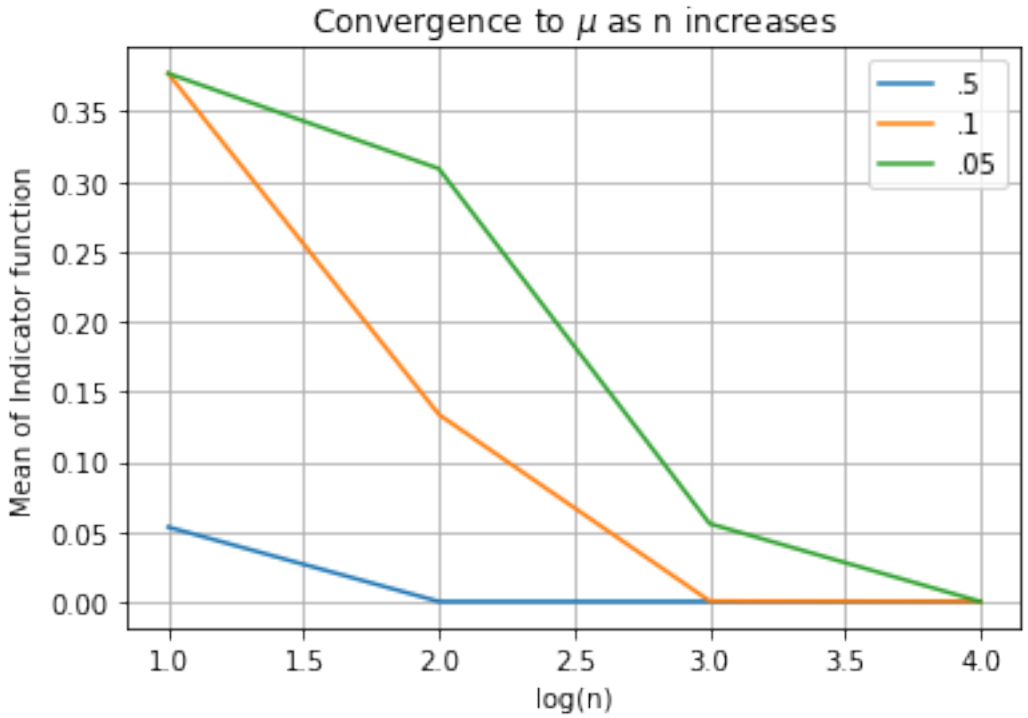
This plot shows the decreasing deviation of $\bar{X}_n^{(1)} - \mu$ as n increases. The absolute deviation is monotonically decreasing as n increases. As n increases, the deviation converges to 0.

1.3.3 (b) Draw $log_{10}(n)$ v.s. $\frac{I}{N}\Sigma_{i=1}^{N}\mathbb{1}\{|\bar{X}_n^{(i)} - \mu| > \epsilon\}$ for $\epsilon = 0.5, \epsilon = 0.1, \epsilon = 0.05$

[11]:
```
def indicator_func(list2check, epsilon):
    n = len(list2check)
    return sum(np.where(list2check > epsilon, 1, 0))/n
```

[18]:
```
epsilon_50 = [indicator_func(results[n,:], .5) for n in range(4)]
epsilon_10 = [indicator_func(results[n,:], .1) for n in range(4)]
epsilon_05 = [indicator_func(results[n,:], .05) for n in range(4)]
```

```
fig, ax = plt.subplots(1, 1)
ax.plot(log_n, epsilon_50)
ax.plot(log_n, epsilon_10)
ax.plot(log_n, epsilon_05)
ax.set(xlabel = "log(n)",
       ylabel = "Mean of Indicator function",
       title = "Convergence to $\mu$ as $n$ increases"
      )
ax.grid()
plt.legend([".5", ".1", ".05"])
plt.show()
```

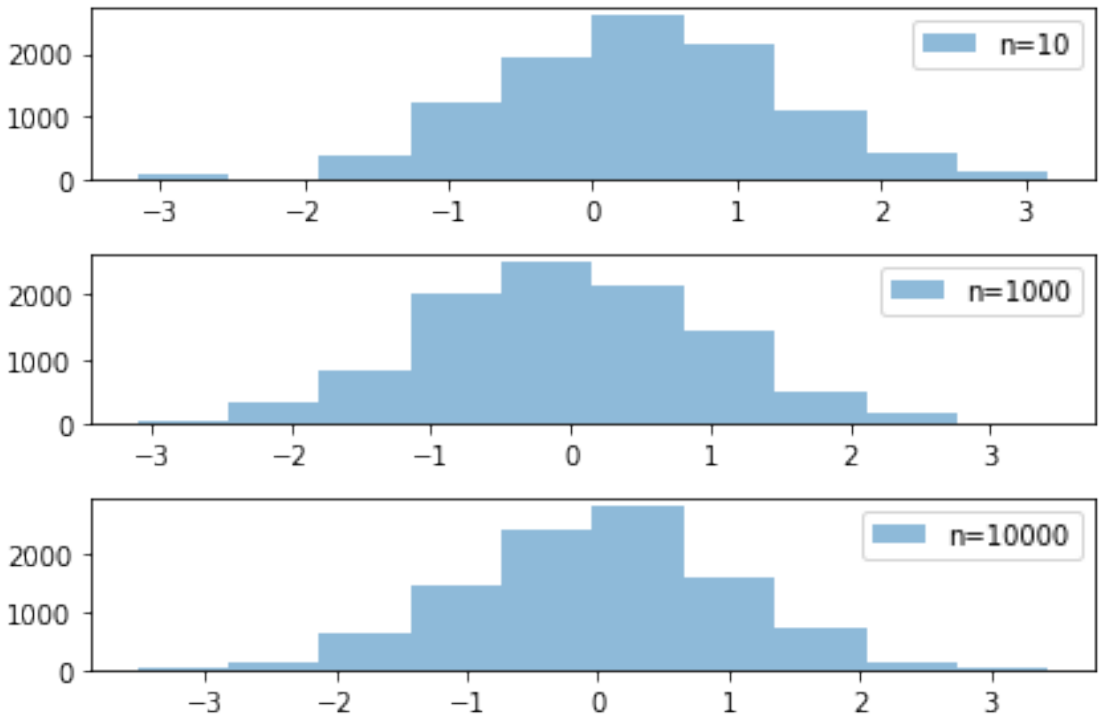


We can view this as an example of the law of large numbers which states, in part, "the average of the results obtained from a large number of trials should be close to the expected value and will tend to become closer to the expected value as more trials are performed". Note, all the trials are monotonic and decreasing as the empiracal averages approach the true expectation.

1.3.4 (c) Draw histograms of $\sqrt{n}(\bar{X}_n^{(i)} - \mu)/\sigma$ for N datasets for $n = 10, n = 1,000, n = 10,000$. You may choose your histogram bins or you may let Python choose automatically–any meaningful plot will do.

[13]:
```
def hist_func(listn, mu, sigma, n):
    return np.sqrt(n)*(listn - mu)/sigma
```

[17]:
```
hist10 = hist_func(results[0, :], 0, 1, 10)
hist1000 = hist_func(results[2, :], 0, 1, 1000)
hist10000 = hist_func(results[3, :], 0, 1, 10000)

fig, ax = plt.subplots(3, 1)

ax[0].hist(hist10,
         alpha=0.5, # the transaparency parameter
         label='n=10')

ax[1].hist(hist1000,
         alpha=0.5,
         label='n=1000')

ax[2].hist(hist10000,
       alpha = 0.5,
       label = 'n=10000')

ax[0].legend(loc='best')
ax[1].legend(loc='best')
ax[2].legend(loc='best')

plt.tight_layout()
plt.show()
```



The histograms, or distributions, of all datasets all exhibit normal distributions. This is expected due to the Central Limit Theorem which states, in part, "when independent random variables are added, their properly normalized sum tends toward a normal distribution (informally a bell curve) even if the original variables themselves are not normally distributed".

1.4 3. Principal Component Analysis

1.4.1 (a) Load all images

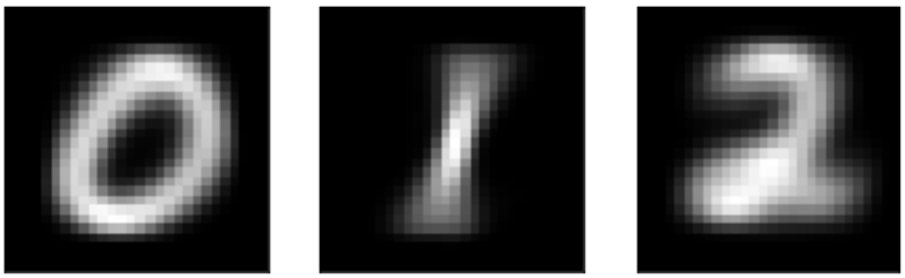
[26]:
```
#https://stackoverflow.com/questions/38675389/
 ↪python-opencv-how-to-load-all-images-from-folder-in-alphabetical-order
filenames = glob.glob("homework1-data/image0/*.pgm")
filenames.sort()
image0 = [cv2.imread(img) for img in filenames]
image0 = np.array(image0)
image0 = image0[:, :, :, 0]

filenames = glob.glob("homework1-data/image1/*.pgm")
filenames.sort()
image1 = [cv2.imread(img) for img in filenames]
image1 = np.array(image1)
image1 = image1[:, :, :, 0]
```

```
filenames = glob.glob("homework1-data/image2/*.pgm")
filenames.sort()
image2 = [cv2.imread(img) for img in filenames]
image2 = np.array(image2)
image2 = image2[:, :, :, 0]

print(np.shape(image0))
print(np.shape(image1))
print(np.shape(image2))
```

```
(5000, 28, 28)
(5000, 28, 28)
(5000, 28, 28)
```

1.4.2 (b) Calculate and plot group means

[27]:
```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3)

mean0 = np.mean(image0, axis = 0)
ax1.imshow(mean0, cmap='gray')

mean1 = np.mean(image1, axis = 0)
ax2.imshow(mean1, cmap='gray')

mean2 = np.mean(image2, axis = 0)
ax3.imshow(mean2, cmap='gray')

ax1.tick_params(left = False, right = False , labelleft = False ,
                labelbottom = False, bottom = False)
ax2.tick_params(left = False, right = False , labelleft = False ,
                labelbottom = False, bottom = False)
ax3.tick_params(left = False, right = False , labelleft = False ,
                labelbottom = False, bottom = False)
plt.show()
```



1.4.3 (c) Form a matrix and standarize

[28]:
```
image_matrix = np.concatenate(
    (image0.reshape((5000, 784)),
     image1.reshape((5000, 784)),
     image2.reshape((5000, 784)),
    )
)
```

[29]:
```
from sklearn.preprocessing import StandardScaler
image_matrix = StandardScaler().fit_transform(image_matrix)
np.shape(image_matrix)
```

[29]:
```
(15000, 784)
```

1.4.4 (d) Perform PCA

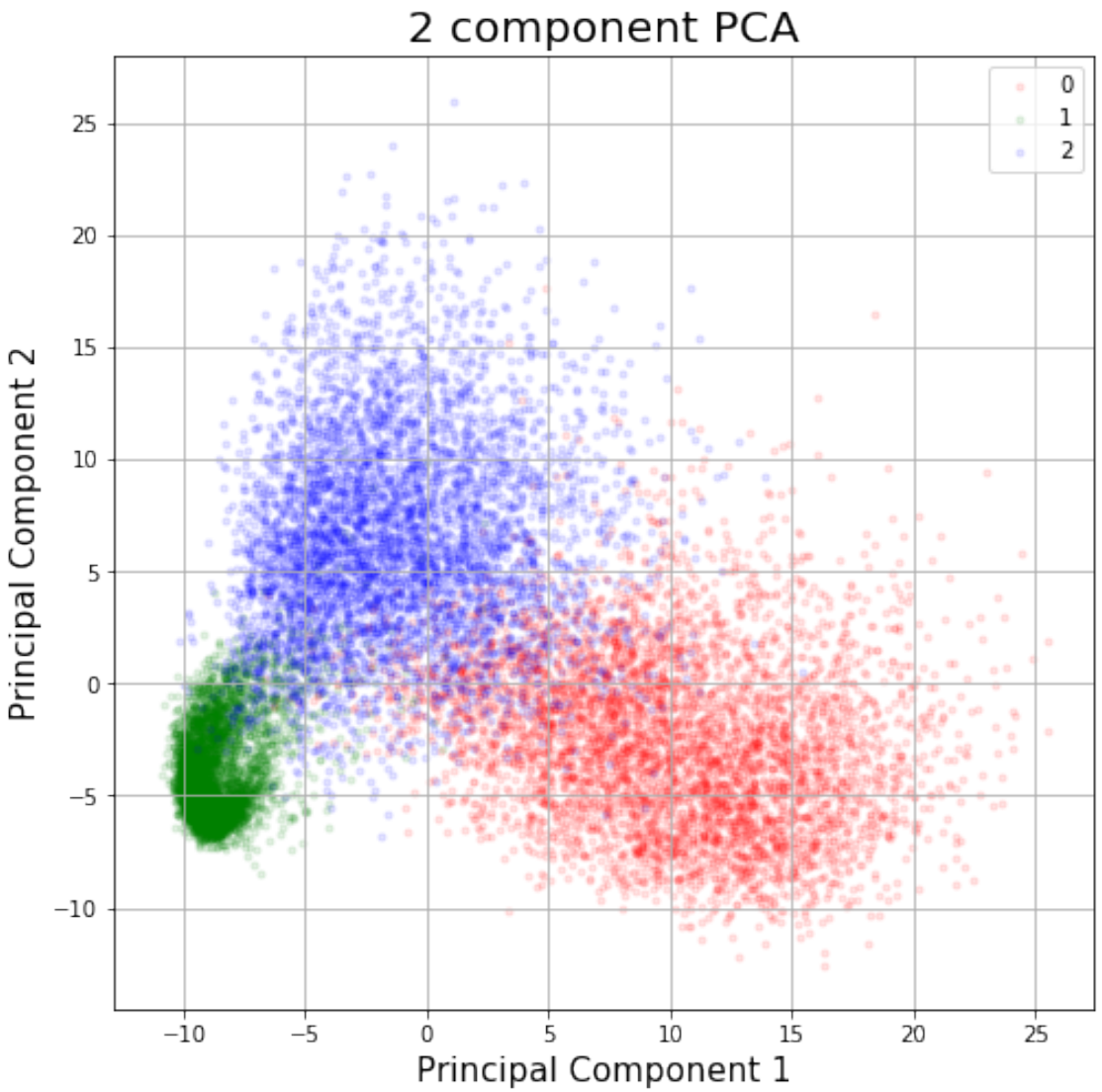
[30]:
```
from sklearn.decomposition import PCA
import pandas as pd
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(image_matrix) ## transforming the data␣
 ↪into PC coordinates.
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])
```

1.4.5 (e) Plot 2 component PCA

[31]:
```
principalDf['number'] = 0
principalDf.iloc[5000:10000, 2] = 1
principalDf.iloc[10000:15000, 2] = 2
```

[32]:
```
targets = [0, 1, 2]
colors = ['r', 'g', 'b']

fig, ax = plt.subplots(figsize = (8,8))
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)

for target, color in zip(targets,colors):
    indicesToKeep = principalDf['number'] == target
    ax.scatter(principalDf.loc[indicesToKeep, 'principal component 1'],
               principalDf.loc[indicesToKeep, 'principal component 2'],
               c = color,
```

```
                s = 10,
                alpha = 0.1
               )

ax.legend(targets)
ax.grid()
plt.show()
```



1.4.6 (f) Observations

There are 3 distinct clusters corresponding to the three digits. Images corresponding to the digit 0 are those most likely to have a principal component 1 value greater than 5 and principal component

value 2 less than 5. Images corresponding to the digit 1 are thsoe most likely to have a principal ocmponent 1 value between -10 and 5, with a principal component 2 value greater than 0. Images corresponding to the digit 2 are those with principal component 1 values between -5 and -10, and a principal ocmponent 2 value between -5 and 0. There exists some crossover between the three digits so we PCA is not a perfect classifier but it appears to give good results.