Práctica 4: Eigenfaces

Aguilar Luna Gabriel Daniel, Rodríguez Agiss Zuriel Uzai

***Abstract*—Practica de la clase de Reconocimiento de Patrones que representa el aprendizaje aprendido en cuánto a Eigenfaces**

***Index Terms*—Eigenfaces, PCA**

I. Objetivo

Aplicar el análisis de componentes principales (PCA) y k-medias al análisis de "caras conocidas".

II. Introduction

El algoritmo de PCA es ampliamente utilizado en una variedad de temas. Usarlo en rostros lo hace más interpretable por humanos, por lo que es una de las aplicaciones más populares. Eigenfaces es un método que es útil para el reconocimiento y la detección de rostros al determinar la variación de rostros en una colección de imágenes de rostros y usar esas variaciones para codificar y decodificar un rostro en una forma de aprendizaje automático sin la información completa, lo que reduce la complejidad del espacio y el cálculo.

Basándonos en $Ax = \lambda x$, casi todos los vectores cambian de dirección cuando se multiplican por A. Ciertos vectores excepcionales x están en el mismo dirección como Ax. Esos son los "vectores propios" (eigenvectors). Multiplique un vector propio por A, y el el vector Ax es un número multiplicado por la x original. La ecuación básica es $Ax = \lambda x$. El número es un "valor propio" (eigenvalue) de A.

El objetivo principal del PCA es la reducción de la dimensionalidad. Tiene muchas aplicaciones en visualización, extracción de características, compresión de datos, etc. La idea detrás de esto es proyectar linealmente los datos originales en un subespacio de menor dimensión ofreciendo los componentes principales (autovectores) la varianza máxima de los datos proyectados y/o el error de distorsión mínimo de la proyección. Eventualmente, ambos conducen al mismo resultado, que es la mejor fórmula de reconstrucción. Como nota al margen, este subespacio se llama subespacio principal.

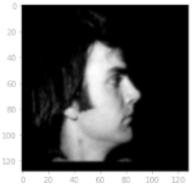
III. Desarrollo

Se tienen imágenes sin extensión. Leyéndolas observamos que tenemos imágenes de este tipo

Dataset

```
In [2]: original_images = []
        basedir = './rawdata/'
        files = []
        lsita_imagenes = [x for x in os.listdir(basedir) if os.path.isfile(os.path.join(basedir, x))]
        i = 0
        while i < 500:
            file = random.choice(lsita_imagenes)
            if file not in files:
                files.append(file)
                with open(basedir+file, 'rb') as image_file:
                    original_images.append(np.array(Image.frombytes("L", (128, 128), image_file.read())))
                i += 1
```

```
In [3]: plt.imshow(original_images[0], cmap='gray')
        plt.show()
```



Lo que sigue es hacer preprocesamiento para estandarizar el tamaño de acuerdo a la altura y anchura de la imagen más pequeña, y obtener la imagen promedio de todo el conjunto

Preprocessing

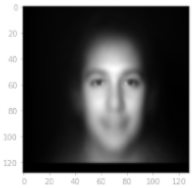
```
In [4]: min_rows, min_cols = sys.maxsize, sys.maxsize
        max_rows, max_cols = 0, 0
        for (i, image) in enumerate(original_images):
            r, c = image.shape[0], image.shape[1]
            min_rows = min(min_rows, r)
            max_rows = max(max_rows, r)
            min_cols = min(min_cols, c)
            max_cols = max(max_cols, c)

        print("\n==> Least common image size:", min_rows, "x", min_cols, "pixels")
```

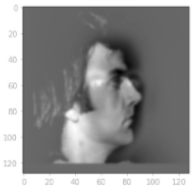
```
==> Least common image size: 128 x 128 pixels
```

```
In [5]: images = np.array(original_images)
        mean_image = np.mean(images, axis=0)
```

```
In [6]: plt.imshow(mean_image, cmap='gray')
        plt.show()
```



```
In [7]: centered_images = np.array([imagen-mean_image for imagen in images])
        plt.imshow(centered_images[0], cmap='gray')
        plt.show()
```



Después de eso, necesitamos realizar la descomposición de valores singulares en los datos centrados para encontrar esos componentes principales llamados ”caras propias” (eigenfaces). Se almacena el resultado en tres matrices, U, Sigma y VT, donde U contiene U, Sigma contiene solo las entradas diagonales de $\sum$ y VT contiene V. La gráfica muestra los valores singulares como puntos, graficados en cada posición $x = i$ para los i -ésimos valores singulares. Para dar una idea aproximada de la rapidez con la que decaen los valores singulares, el gráfico incluye una línea sólida que muestra la curva, $\sigma 0/(i+1)$.

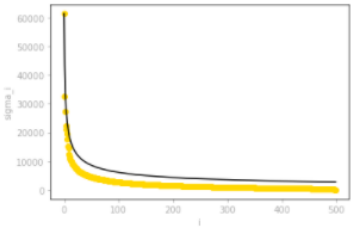
Applying PCA

```
U, Sigma, VT = np.linalg.svd(X, full_matrices=False)
```

```
# Sanity check on dimensions
print("X:", X.shape)
print("U:", U.shape)
print("Sigma:", Sigma.shape)
print("V^T:", VT.shape)
```

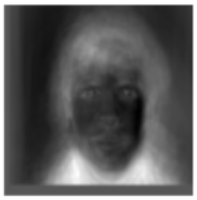
```
X: (500, 16384)
U: (500, 500)
Sigma: (500,)
V^T: (500, 16384)
```

```
plt.scatter(range(len(Sigma)),Sigma, c='gold')
plt.plot(range(len(Sigma)),[Sigma[0]/((i+1)**0.5) for i in range(len(Sigma))], c='black')
plt.xlabel('i', c='darkgray')
plt.ylabel('sigma_i', c='darkgray')
plt.show()
```



Se procede a calcular matriz Y, que es la matriz de datos original proyectada sobre los primeros componentes principales num_components.

```
num_components = 5 # Number of principal components
for compo in range(num_components):
    plt.imshow(np.reshape(VT[compo, :],(min_rows, min_cols)), cmap='gray')
    plt.axis('off')
    plt.show()
```



A. *Cálculo de Eigenfaces*

La idea principal detrás de las Eigenfaces ests la siguiente: Se supone que γ es un vector de N^2x1, correspondiente a una imagen de NxN La idea es representar $\gamma(\phi = \gamma - imagenpromedio)enunespaciodemenordimensionalidad$

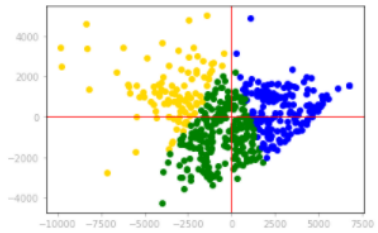
$$\phi - mean = w_1u_1 + w_2u_2 + ...w_ku_k(K << N^2)$$

Se procede a ejecutar k -means en los datos proyectados, Y [: m,: num _components], para intentar identificar hasta num clusters clústeres.

```
modelo_kmeans = KMeans(n_clusters=3).fit(np.dstack((Y[:,0],Y[:,1]))[0])
```

```
clasiKmeans = modelo_kmeans.predict(np.dstack((Y[:,0],Y[:,1]))[0])
```

```
plt.axline((0, 0), (0, 1), linewidth=1, color='r')
plt.axline((0, 0), (1, 0), linewidth=1, color='r')
colores = ['gold','green','blue']
for i,k in enumerate(clasiKmeans):
    plt.scatter(Y[i,0],Y[i,1], c=colores[k])
plt.show()
```



Paso 1: Se deben obtener las imágenes de entrenamiento $I_1, I_2, ..., I_M$ previamente preprocesadas y del mismo tamaño

```
corpus = []
basedir = './lfw1000/'
files = []
lsita_imagenes = [x for x in os.listdir(basedir) if os.path.isfile(os.path.join(basedir, x))]
i = 0
while i < 1000:
    file = random.choice(lsita_imagenes)
    if file not in files:
        files.append(file)
        corpus.append(np.array(Image.open(basedir+file)))
        i += 1
```

```
plt.imshow(corpus[0], cmap='gray')
plt.show()
```

Paso 2: Se representa cada imagen I_i como un vector Γ_i

```
min_rows, min_cols = sys.maxsize, sys.maxsize
max_rows, max_cols = 0, 0
for (i, image) in enumerate(corpus):
    r, c = image.shape[0], image.shape[1]
    min_rows = min(min_rows, r)
    max_rows = max(max_rows, r)
    min_cols = min(min_cols, c)
    max_cols = max(max_cols, c)

Images = np.array(corpus)
```

```
# Create m x d data matrix
m = len(Images)
d = min_rows * min_cols
Gamma = np.reshape(Images, (m, d))
```

```
# Sanity check on dimensions
print("Gamma:", Gamma.shape)
```

```
Gamma: (1000, 4096)
```

Paso 3. Se calcula el vector promedio Ψ : $\Psi = \frac{1}{M}\sum_{i=1}^{M}\Gamma_i$

```
Psi = np.mean(Gamma, axis=0)
```

Paso 4: Se sustrae la cara promedio $\Phi = \Gamma_i - \Psi$

```
Phi = np.array([imagen-Psi for imagen in Gamma])
Phi.shape
```

```
(1000, 4096)
```

Paso 5: Se calcula la matrix de covarianza C:

$$C = \frac{1}{M}\sum_{n=1}^{M}\Phi_n\Phi_n^T = AA^T(matrizN^2xN^2)$$

$$dondeA = [\Phi_1, \Phi_2 ... \Phi_M](matrizN^2xM)$$

```
C = np.transpose(Phi).dot(Phi)
C.shape
```

```
(4096, 4096)
```

Paso 6: Se calculan los eigenvectores u_i de AA^T

```
# AA^T is very large
# Consider the matrix A^TA
ATA = Phi.dot(np.transpose(Phi))
ATA.shape
```

```
(1000, 1000)
```

```
# compute the eigenvectors vi of ATA
W, V = np.linalg.eig(ATA)

print("W:", W.shape)
print("V:", V.shape)
```

```
W: (1000,)
V: (1000, 1000)
```

```
# compute the eigenvectors ui of AAT
UU, U = np.linalg.eig(C)

print("UU:", UU.shape)
print("U:", U.shape)
```

```
UU: (4096,)
U: (4096, 4096)
```

Paso 7: Se mantienen solo K eigenvectores (correspondientes a los K eigenvalores más grandes)

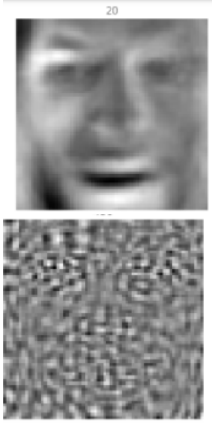
```
K = 1000
```

La meanface (cara promedio) resultante es la siguiente

```
plt.imshow(np.reshape(Psi,(min_rows, min_cols)), cmap='gray')
plt.axis('off')
plt.show()
```

El proceso de convertir a eigenfaces es el siguiente

```
for k in range(20,500,20):
    plt.title(str(k), fontdict={'color': [.5,.5,.5]})
    plt.imshow(np.reshape(np.real(U[:,k]),(min_rows, min_cols)), cmap='gray')
    plt.axis('off')
    plt.show()
```



IV. Resultado

Este es solo un ejemplo de los resultados obtenidos. Se va a intentar reconstruir la siguiente imagen

```
imagen_prueba = findImage()
```

```
(64, 64)
```

Se obtiene el siguiente resultado.

```
reconstruction(imagen_prueba)
```

```
array([152.32122658+0.j, 158.28941079+0.j, 170.29670321+0.j, ...,
       166.50374061+0.j, 162.71549536+0.j, 169.73898706+0.j])
```

V. Código

```
import sys
import numpy as np
import os, random
import matplotlib.pyplot as plt
import matplotlib.image as pltI
from PIL import Image
from sklearn.cluster import KMeans
from IPython.display import HTML
plt.rc('xtick',color='darkgray')
plt.rc('ytick',color='darkgray')

def gif_maker(basedir = './reco/', nombre='reco'):
    gif_frames = []
    lista_imagenes = [x for x in os.listdir(basedir)
     if os.path.isfile(os.path.join(basedir, x))]
    for i in lista_imagenes:
        new_frame = Image.open(basedir+i)
        gif_frames.append(new_frame)
    gif_frames[0].save(nombre+'.gif', format='GIF',
    append_images=gif_frames[1:], save_all=True,
    duration=300, loop=0)

def findImage():
    find = True
    while find:
        file = random.choice(lsita_imagenes)
        if file not in files:
            files.append(file)
            imagen_prueba = np.array(Image.open(
    basedir+file))
            print(imagen_prueba.shape)
            find = False
    plt.imshow(imagen_prueba, cmap='gray')
    plt.axis('off')
    plt.show()
    return imagen_prueba

def reconstruction(imagen):
    X = np.reshape(imagen, (d))
    Y = np.matmul(X, U[:,:1001])
    aprox_image = Y[0]*U[:,0]
    mostrar =
    [1,5,10,50,100,150,200,300,500,700,1000]
    for i in range(1,1001):
        aprox_image += Y[i]*U[:,i]
        if i in mostrar:
            plt.title(str(i), fontdict={'color':
    [.5,.5,.5]})
            plt.imshow(np.reshape(np.real(
    aprox_image),(min_rows, min_cols)), cmap='gray')
            plt.axis('off')
            plt.show()
        if (i%20 == 0) or (i<100 and i%10==0) or (i
    <10 and i%5==0):
            pltI.imsave('reco/i_'+str(bin(i))[2:].
    zfill(10)+'.png', np.reshape(np.real(aprox_image
    ),(min_rows, min_cols)), cmap='gray')
        pltI.imsave('reco/z.png', np.reshape(np.real
    (imagen),(min_rows, min_cols)), cmap='gray')
    return aprox_image

original_images = []
basedir = './rawdata/'
files = []
lsita_imagenes = [x for x in os.listdir(basedir) if
    os.path.isfile(os.path.join(basedir, x))]
i = 0
while i < 500:
    file = random.choice(lsita_imagenes)
    if file not in files:
        files.append(file)
        with open(basedir+file, 'rb') as image_file:
            original_images.append(np.array(Image.
    frombytes("L", (128, 128), image_file.read())))
        i += 1

plt.imshow(original_images[0], cmap='gray')
plt.show()

min_rows, min_cols = sys.maxsize, sys.maxsize
max_rows, max_cols = 0, 0
for (i, image) in enumerate(original_images):
    r, c = image.shape[0], image.shape[1]
    min_rows = min(min_rows, r)
    max_rows = max(max_rows, r)
    min_cols = min(min_cols, c)
    max_cols = max(max_cols, c)

print("\n==> Least common image size:", min_rows, "x
    ", min_cols, "pixels")

images = np.array(original_images)
mean_image = np.mean(images, axis=0)

plt.imshow(mean_image, cmap='gray')
plt.show()

centered_images = np.array([imagen-mean_image for
    imagen in images])
plt.imshow(centered_images[0], cmap='gray')
plt.show()

# Create m x d data matrix
m = len(images)
d = min_rows * min_cols
X = np.reshape(centered_images, (m, d))

U, Sigma, VT = np.linalg.svd(X, full_matrices=False)

# Sanity check on dimensions
print("X:", X.shape)
print("U:", U.shape)
print("Sigma:", Sigma.shape)
print("V^T:", VT.shape)

plt.scatter(range(len(Sigma)),Sigma, c='gold')
plt.plot(range(len(Sigma)),[Sigma[0]/((i+1)**0.5)
    for i in range(len(Sigma))], c='black')
plt.xlabel('i', c='darkgray')
plt.ylabel('sigma_i', c='darkgray')
plt.show()
```

```
num_components = 5 # Number of principal components
for compo in range(num_components):
    plt.imshow(np.reshape(VT[compo, :],(min_rows, min_cols)), cmap='gray')
    plt.axis('off')
    plt.show()

Y = np.matmul(X, VT[:num_components,:].T)
Y.shape

plt.axline((0, 0), (0, 1), linewidth=1, color='r')
plt.axline((0, 0), (1, 0), linewidth=1, color='r')
plt.scatter(Y[:,0],Y[:,1], c='forestgreen')
plt.show()

modelo_kmeans = KMeans(n_clusters=3).fit(np.dstack((Y[:,0],Y[:,1]))[0])

clasiKmeans = modelo_kmeans.predict(np.dstack((Y[:,0],Y[:,1]))[0])

plt.axline((0, 0), (0, 1), linewidth=1, color='r')
plt.axline((0, 0), (1, 0), linewidth=1, color='r')
colores = ['gold','green','blue']
for i,k in enumerate(clasiKmeans):
    plt.scatter(Y[i,0],Y[i,1], c=colores[k])
plt.show()

# Computing the EigenFaces

corpus = []
basedir = './lfw1000/'
files = []
lsita_imagenes = [x for x in os.listdir(basedir) if os.path.isfile(os.path.join(basedir, x))]
i = 0
while i < 1000:
    file = random.choice(lsita_imagenes)
    if file not in files:
        files.append(file)
        corpus.append(np.array(Image.open(basedir+file)))
        i += 1
plt.imshow(corpus[0], cmap='gray')
plt.show()

min_rows, min_cols = sys.maxsize, sys.maxsize
max_rows, max_cols = 0, 0
for (i, image) in enumerate(corpus):
    r, c = image.shape[0], image.shape[1]
    min_rows = min(min_rows, r)
    max_rows = max(max_rows, r)
    min_cols = min(min_cols, c)
    max_cols = max(max_cols, c)

Images = np.array(corpus)

# Create m x d data matrix
m = len(Images)
d = min_rows * min_cols
Gamma = np.reshape(Images, (m, d))

# Sanity check on dimensions
print("Gamma:", Gamma.shape)

Psi = np.mean(Gamma, axis=0)

Phi = np.array([imagen-Psi for imagen in Gamma])
Phi.shape

C = np.transpose(Phi).dot(Phi)
C.shape

# AA^T is very large
# Consider the matrix A^TA
ATA = Phi.dot(np.transpose(Phi))
ATA.shape

# compute the eigenvectors vi of ATA
W, V = np.linalg.eig(ATA)

print("W:", W.shape)
print("V:", V.shape)

# compute the eigenvectors ui of AAT
UU, U = np.linalg.eig(C)

print("UU:", UU.shape)
print("U:", U.shape)

K = 1000

plt.imshow(np.reshape(Psi,(min_rows, min_cols)), cmap='gray')
plt.axis('off')
plt.show()

for k in range(10):
    plt.title(str(k), fontdict={'color': [.5,.5,.5]})
    plt.imshow(np.reshape(np.real(U[:,k]),(min_rows, min_cols)), cmap='gray')
    plt.axis('off')
    plt.show()


for k in range(20,500,20):
    plt.title(str(k), fontdict={'color': [.5,.5,.5]})
    plt.imshow(np.reshape(np.real(U[:,k]),(min_rows, min_cols)), cmap='gray')
    plt.axis('off')
    plt.show()
imagen_prueba = findImage()
reconstruction(imagen_prueba)
HTML('<img src="./reco4.gif" width="300">')
alexa = np.array(Image.open('imagenes/alexa.png'))
plt.imshow(alexa, cmap='gray')
plt.axis('off')
plt.show()

reconstruction(alexa)

wombat = np.array(Image.open('imagenes/wombat.jpg'))
plt.imshow(wombat, cmap='gray')
plt.axis('off')
plt.show()
reconstruction(wombat)
```

VI. Conclusiones

Esta práctica nos sirvió para darnos cuenta de lo relativamente fácil que es implementar conceptos como eigenvectores o eigenvalores para reconstruir imágenes faciales a partir de información clave de forma que se reduce la cantidad de información que se tiene que almacenar. Por ende, los objetivos de la práctica fueron cumplidos

VII. Referencias

Acar, N. (2018) Eigenfaces: Recovering Humans from Ghosts. Consultado de https://towardsdatascience.com/eigenfaces-recovering-humans-from-ghosts-17606c328184

Dey., S. (2018) EigenFaces and A Simple Face Detector with PCA/SVD in Python, sandipanweb. Consultado en

https://sandipanweb.wordpress.com/2018/01/06/eigenfaces-and-a-simple-face-detector-with-pca-svd-in-python/. [Accessed: 02- Aug- 2021].

Strang, G. (2016) Introduction to Linear Algebra. Capítulo 6. Eigenvalues and Eigenvectors. MIT. Consultado de http://math.mit.edu/ gs/linearalgebra/ila0601.pdf