Tarea 12: eigenfaces y fisherfaces
Leonardo CA

July 16, 2020

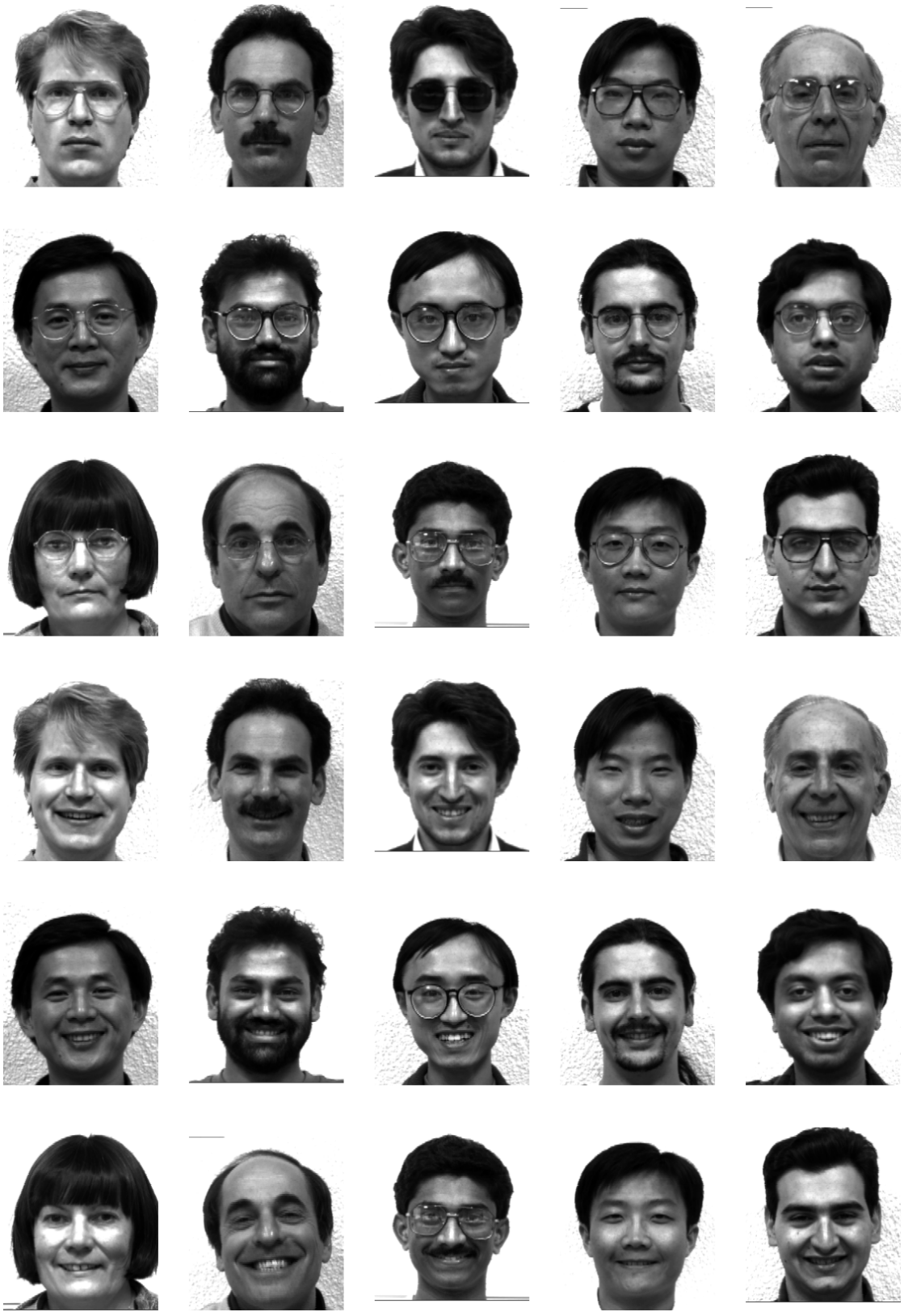
Primero cargamos la base de datos de caras de YALE, que tiene 165 fotografías de 15 sujetos diferentes haciendo 11 expresiones faciales diferentes (feliz, triste, sonriendo, guiño, entre otros). De ante mano se agruparon las imágenes en estas clases.

[2]:
```
import numpy as np

#pattern=['_centerlight','_glasses','_happy',␣
↪'leftlight','_noglasses','_normal','_rightlight','_sad','_sleepy','_surprised','_sleepy']

from PIL import Image
import os, os.path
yale=[]
path ='/home/noxd/datos/faces/YALE/centered_1/'
valid_images = [".jpg",".gif",".png",".tga"]
for f in sorted(os.listdir(path)):
    ext = os.path.splitext(f)[1]
    if ext.lower() not in valid_images:
        continue
    yale.append(np.array(Image.open(os.path.join(path,f))))
```

Graficamos algunos de los sujetos (los primeros 3, con sus diferentes expresiones, ocasionalmente usan lentes).

[5]:
```
(45045, 165)
```

[54]:
```
# Crea figura y ejes (en este caso es un arreglo de 3 x 3)
fig=plt.figure(figsize=(20, 200))
# Hace los ejes accesibles con una sola indexación
columns = 5
rows = 33
for i in range(1,cte):
fig.add_subplot(rows, columns, i)
plt.imshow(yale[i],cmap='gray')
plt.axis('off')
```



Seguido se aplanan las imágenes en su forma vectorial y los agrupamos todos en una matriz Q.

[5]:
```
nr=yale[0].shape[0]
nc=yale[0].shape[1]
# numero de variables mas uno
n=nr*nc
cte=int(len(yale))
Q= np.zeros(shape=(n,cte))
Q_array=np.zeros(shape=(n,1))
med= np.zeros(shape=(cte,cte))
for i in range(cte):
    Q_array = yale[i].flatten()  # covert 2d to 1d array
    Q[:,i] = Q_array
Q.shape
```

Se calcula la imagen "media" que es el vector que promedia los valores de la matriz Q. También se aprovecha el ciclo para hacer una matriz $Qsinm$ en la cual se le resta los valores de la cara "media" a todas imágenes.

[9]:
```
media=np.zeros(shape=(n,1))
#imagenes sin media
Q_sinm= np.zeros(shape=(n,cte))

for c in range(0,Q.T.shape[1]):
    media[c]= np.mean(Q[c:c+1])
    Q_sinm[c:c+1]=Q[c:c+1]-np.mean(Q[c:c+1])
```

Calculamos la matriz de covarianza de la matriz Q sin media y resolvemos la ecuación $QsnmU = \lambda u$ donde u son los eigenvectores y λ los eigenvalores.

[10]:
```
X=np.cov(Q_sinm.T)
from numpy import linalg as LA
evalues,evector =LA.eig(X)
```

Creamos una variable que indexe el orden de la magnitud de los eigenvalores y lo usamos para ordenar los eigenvalores y eigenvectores de acuerdo con la magnitud de los eigenvalores.

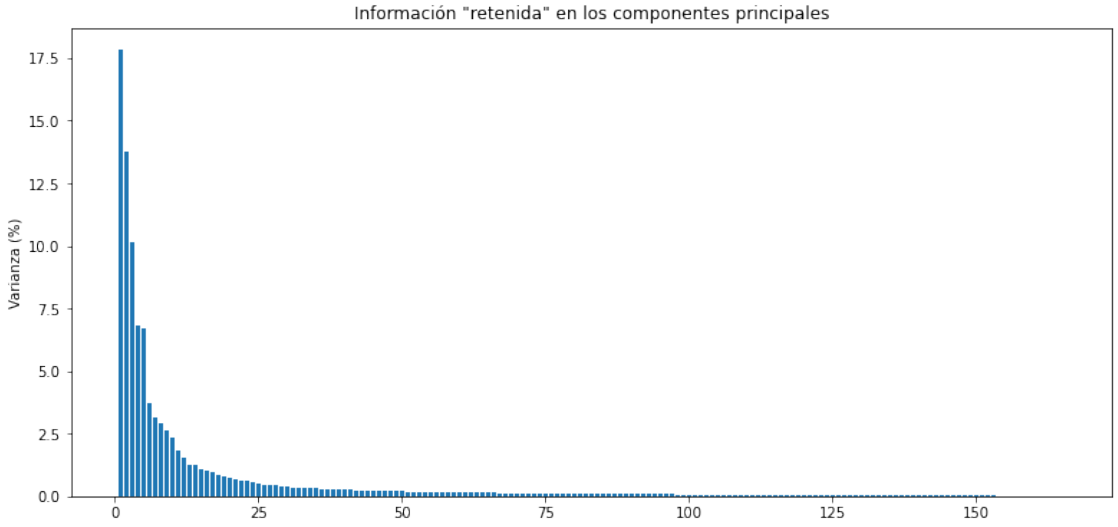
[11]:
```
order = evalues.argsort()[::-1]
evalues1 = evalues[order]
evector1 = evector[:,evalues1.argsort()[::-1]]
```

Se grafica la importancia o peso relativo de los eigenvalores con respecto a su total, se ve que el primer eigenvalor tiene el mayor peso y este disminuye conforme aumenta el orden.

[13]:
```
import matplotlib.pyplot as plt
lista1=[]
lista2=[]
for i in range(0,cte):
```

```
    lista1.append(i+1)
    lista2.append('PC' + str(i+1))

plt.figure(figsize=(13,6))
plt.bar(lista1,evalues1/sum(evalues1)*100,align='center')
plt.ylabel('Varianza (%)')
plt.title('Información "retenida" en los componentes principales')
```

[13]:
```
Text(0.5, 1.0, 'Información "retenida" en los componentes principales')
```



Se genera la matriz de eigenvectores que se usa para transformar la matriz de imágenes original Q en las eigenfaces.

[12]:
```
            ev=np.zeros(shape=(cte,cte-11))
            for l in range(0,cte-11):
            ev.T[0:cte-11,l:l+1]= (1/np.sqrt(evalues1[l]*cte))*evector1.T[0:
↪cte-11,l:l+1]
```

```
/home/noxd/miniconda3/lib/python3.7/site-packages/ipykernel_launcher.py:3:
ComplexWarning: Casting complex values to real discards the imaginary part
This is separate from the ipykernel package so we can avoid doing imports
until
```

Se calculan los eigenfaces ($Yfit$)

[14]:
```
Y_fit=np.matmul(Q,ev)
#Y_fit=Y_fit+media
```

Se transforman las eigen faces en imagenes (matrices) y se guardan en una lista.

[16]:
```
imagenes=[]
result=[]
for e in range(0,cte-11):
    #print(e)
    #c1=np.zeros(shape=(n,1))
    c1=Y_fit[0:n,e:e+1]
    imagenes.append(c1)
    result.append(np.reshape(imagenes[e],[nr,nc]))
    #maxe=np.amax(result[e])
    #mine=np.amin(result[e])
    #for j in range(0,nr):
     #   for i in range(0,nc):
      #      result[e][j,i]=((result[e][j,i]-mine)*(255/(maxe-mine)))
```

Se grafican 30 eigenfaces, 1 cada 5 imágenes, es decir la imagen 1, 5, 10, etcétera. Porque de otro modo me costaba trabajo identificar diferencias.

[51]:
```
# Crea figura y ejes (en este caso es un arreglo de 3 x 3)
fig=plt.figure(figsize=(20, 200))
# Hace los ejes accesibles con una sola indexación
columns = 5
rows = 32
cte1=cte-11

for i in range(0,cte1,5):
    fig.add_subplot(rows, columns, i)
    plt.imshow(result[i],cmap='gray')
    plt.axis('off')
```

En general me costó mucho trabajo ver cuáles son las eigenfaces que corresponden a cada expresión, pero si se alcanzan a distinguir la imagen normal (fila 1 columna 1), feliz (fila 2 columna 4), enojado (fila 1 columna 3), con lentes (fila 1 columna 5), sorpresa (fila 3 columna 1), entre otras

0.1 Fisherfaces

Esta sección es para las Fisherfaces o distinción de caras usando el discriminante lineal de Fisher.

[25]:
```
#Q son las variables, Q_sinm las variables sin medias,
#Y_fit es la matriz de w_a de PCA, y media la media las imagenes
#numero de imagenes per clase
num_cla=15
cte1=cte-15
clases=int(Q.T.shape[0]/num_cla)
```

En esta parte se calcularon las imágenes promedio de cada clase (lentes, sorpresa, feliz, etcétera)

[26]:
```
#medias de clases
m_c=np.zeros(shape=(n,clases))
m_c.shape

for t in range(0,cte1,num_cla):
    tc=int(t/11)
    for c in range(t,t+num_cla):
        #print(t,c,tc)
        m_c.T[tc:tc+1]=Q.T[c:c+1]+m_c.T[tc:tc+1]
m_c=m_c/num_cla
```

La matriz de variación entre clases se realizó restándole la imagen media a la imagen media de cada clase que está siendo multiplicada por la transpuesta de esta misma operación, para disminuir la carga computacional se multiplico de ambos lados por la Wpca (transpuesta del lado izquierdo).

[27]:
```
#between-class scatter matrix be defined as
sb=np.zeros(shape=(cte1,cte1))
for r in range(0,clases-1):
    sb=sb+ np.matmul(Y_fit.T, np.matmul(m_c[0:n,r:r+1]-media,  np.matmul(m_c.T[r:
 ↪r+1] -media.T ,Y_fit) ))
sb=sb*n
```

La matriz de desviación dentro de cada clase se realizó de una forma similar a la matriz anterior, pero se le resta la media de cada clase a la imagen de cada clase que se multiplica por su transpuesta y por Wpca de ambos lados.

[28]:
```
#within-class scatter matrix be defined as
sw=np.zeros(shape=(cte1,cte1))
for t in range(0,cte1,num_cla):
    tc=int(t/11)
    for c in range(t,t+num_cla):
        sw=sw+ np.matmul(Y_fit.T, np.matmul( Q[0:n,c:c+1]-m_c[0:n,tc:tc+1] ,  np.
 ↪matmul(Q.T[c:c+1] -m_c.T[tc:tc+1] ,Y_fit) ))
```

Se multiplica la inversa de la matriz de desviación entre clase por la matriz de desviaciones entre clases.

[29]:
```
#w fisher
wf= np.matmul(np.linalg.inv(sw), sb)
```

La matriz resultado se resuelve para obtener los eigenvectores y eigenvalores de la ecuación, los cuales se denominan Fishervalores y Fishervectores.

[34]:
```
fvalues,fvector =LA.eig(wf)
#fvalues=fvalues.real
#fvector=fvector.real
```

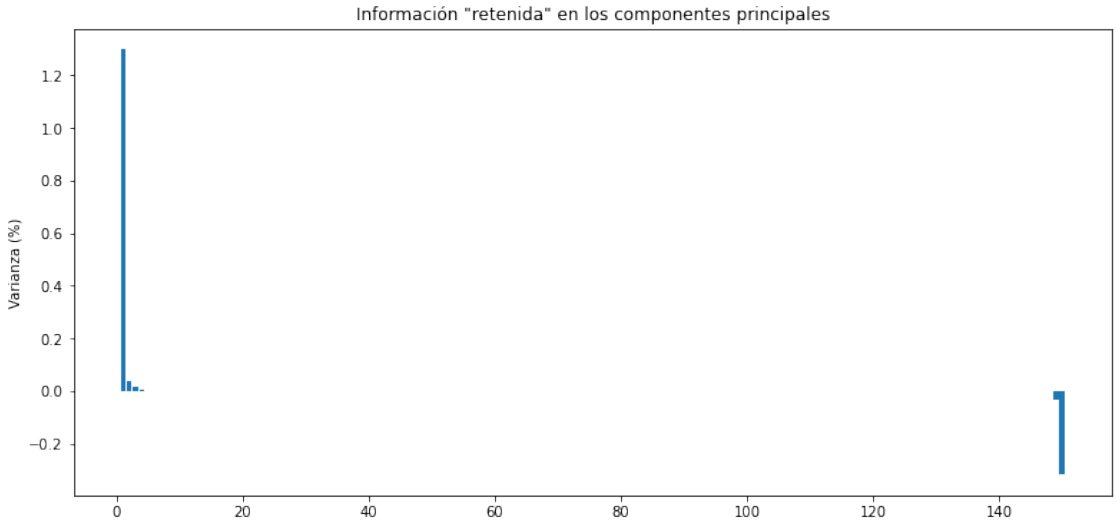
Se ordenan de acuerdo a la magnitud de los fishervalores (tanto los fishervalores como los fishervectores).

[35]:
```
order = fvalues.argsort()[::-1]
fvalues1 = fvalues[order]
fvector1 = fvector[:,fvalues.argsort()[::-1]]
```

Grafique la importancia de los fishervalores y el resultado me apareció diferente, con el ultimo fishervalor teniendo una magnitud relativamente alta en magnitud, pero negativa. No estoy del todo seguro porque sucede pero creo que es por la importancia que toma la parte imaginaria.

[42]:
```
lista1=[]
for i in range(0,Y_fit.shape[1]):
    lista1.append(i+1)
plt.figure(figsize=(13,6))
plt.bar(lista1,fvalues1/sum(fvalues1),align='center')
plt.ylabel('Varianza (%)')
plt.title('Información "retenida" en los componentes principales')
```

[42]:
```
Text(0.5, 1.0, 'Información "retenida" en los componentes principales')
```



Pondere los fishervectores de acuerdo al inverso de la raíz cuadrada de su respectivo fishervalor por 150.

[365]:
```
            cte2=cte1
            ef=np.zeros(shape=(cte1,cte2))
            for l in range(0,cte1):
            ef.T[0:cte1,l:l+1]= (1/np.sqrt(fvalues1[l]*(cte2)))*fvector1.T[0:
↪cte2,l:l+1]
```

Se crean las Fisherfases, multiplicando de nuevo por la matriz Wpca.

[43]:
```
fisherfaces=np.matmul(Y_fit, fvector.real)
```

Se ponen las Fisherfaces como imágenes y se guardan en una lista.

[44]:
```
imagenes=[]
result1=[]
for e in range(0,cte1):
    c1=fisherfaces[0:n,e:e+1]
    imagenes.append(c1)
    result1.append(np.reshape(imagenes[e],[nr,nc]))
    #maxe=np.amax(result[e])
    #mine=np.amin(result[e])
    #for j in range(0,nr):
        #for i in range(0,nc):
            #result[e][j,i]=((result[e][j,i]-mine)*(255/(maxe-mine)))
```

Similar que con las eigenfaces, se grafican 30 fisherfaces, 1 cada 5 imágenes, es decir la imagen 1, 5, 10, etcétera. Porque de otro modo me costaba trabajo identificar diferencias.

[55]:
```
# Crea figura y ejes (en este caso es un arreglo de 3 x 3)
fig=plt.figure(figsize=(20, 200))
# Hace los ejes accesibles con una sola indexación
columns = 5
rows = 32
for i in range(1,cte1):
    fig.add_subplot(rows, columns, i)
    plt.imshow(result1[i],cmap='gray')
    plt.axis('off')
```

Creo que aquí es más sencillo distinguir diferentes expresiones, por ejemplo la imagen en la fila 1 y columna 1 parece adormecido, en la fila 3 y columna 2 parece feliz, en la fila 4 y columna 2 parece sorprendido, en la fila 1 y columna 4 se ve molesto, también hay diversas imágenes con lentes, entre otras.