4-visualizacion

February 16, 2021

#

Ciencia de Datos

Víctor Muñiz Sánchez

Maestría en Cómputo Estadístico

Enero a junio 2021

1 Análisis de Componentes Principales (PCA)

1.1 PCA como un método de reducción de dimensión

1.1.1 Ejemplo 1: Veamos una versión simplificada de los dígitos MNIST

[33]:
```
import numpy as np
import matplotlib.pylab as plt
from sklearn.datasets import load_digits
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
import os

os.chdir('/home/victor/cursos/ciencia_de_datos_2021/')
%matplotlib inline
```

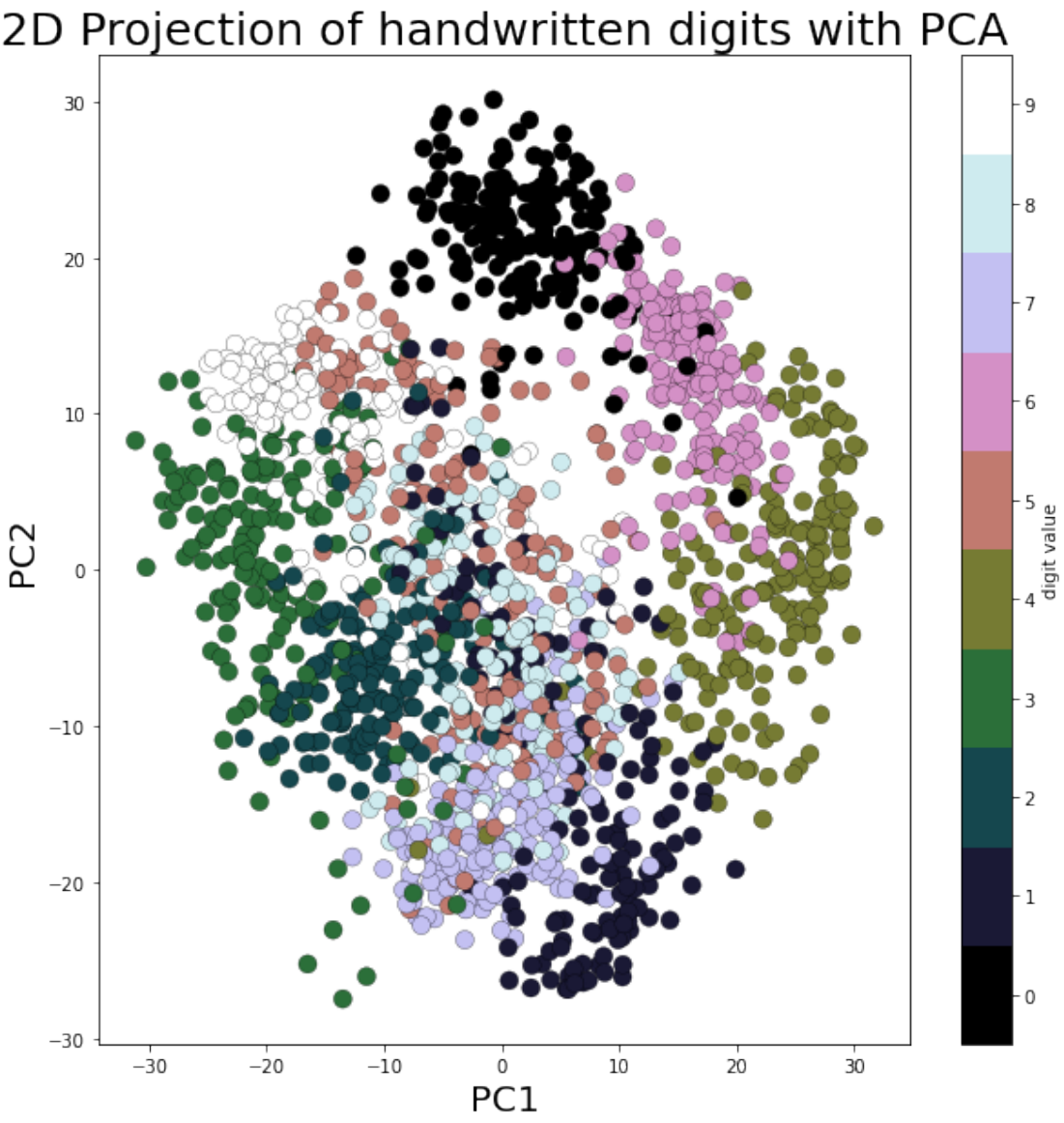
[2]:
```
digits = load_digits()
print('dimensiones:',digits.data.shape)
j = 1
#np.random.seed(1)
fig = plt.figure(figsize=(3,3))
fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.05, wspace=0.05)
for i in np.random.choice(digits.data.shape[0], 25):
    plt.subplot(5,5,j), plt.imshow(np.reshape(digits.data[i,:], (8,8)),␣
 ↪cmap='binary'), plt.axis('off')
    j += 1
plt.show()
```

```
dimensiones: (1797, 64)
```

PCA

[3]:
```
pca_digits=PCA(2)
digits.data_proj = pca_digits.fit_transform(digits.data)
print(np.sum(pca_digits.explained_variance_ratio_))
plt.figure(figsize=(10,10))
plt.scatter(digits.data_proj[:, 0], digits.data_proj[:, 1], lw=0.25, c=digits.
 ↪target, edgecolor='k',  s=100, cmap=plt.cm.get_cmap('cubehelix', 10))
plt.xlabel('PC1', size=20), plt.ylabel('PC2', size=20), plt.title('2D␣
 ↪Projection of handwritten digits with PCA', size=25)
plt.colorbar(ticks=range(10), label='digit value')
plt.clim(-0.5, 9.5)
```

```
0.2850936482369005
```



1.1.2 Ejemplo 2: Eigenfaces

[24]:
```
from sklearn.datasets import fetch_olivetti_faces
faces = fetch_olivetti_faces().data
print('dimensiones:',faces.shape)

fig = plt.figure(figsize=(10,10))
fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.05, wspace=0.05)
# plot 25 random faces
j = 1
np.random.seed(0)
```

```
for i in np.random.choice(range(faces.shape[0]), 25):
    ax = fig.add_subplot(5, 5, j, xticks=[], yticks=[])
    ax.imshow(np.reshape(faces[i,:],(64,64)), cmap=plt.cm.gray,␣
 ↪interpolation='nearest')
    j += 1
plt.show()
```

```
dimensiones: (400, 4096)
```

[34]:
```
# guardo las imagenes como png desde arreglos numpy
for i in range(faces.shape[0]):
    plt.imsave('data/faces/faces_64/img'+str(i+1)+'.png',
               np.reshape(faces[i,:],(64,64)),cmap='gray')
```

Rostros centrados y su varianza ¿Qué pasa si no se estandarizan?

[26]:
```
scaler = StandardScaler(with_mean=True,with_std=True)
faces_scale = scaler.fit_transform(faces)
mean_face = np.reshape(scaler.mean_, (64,64))
sd_face = np.reshape(np.sqrt(scaler.var_), (64,64))

plt.figure(figsize=(10,5))
plt.subplot(121), plt.imshow(mean_face, cmap=plt.cm.gray), plt.axis('off'), plt.
↪title('Mean face')
plt.subplot(122), plt.imshow(sd_face, cmap=plt.cm.gray), plt.axis('off'), plt.
↪title('SD face')
plt.show()
faces_scale.shape
```

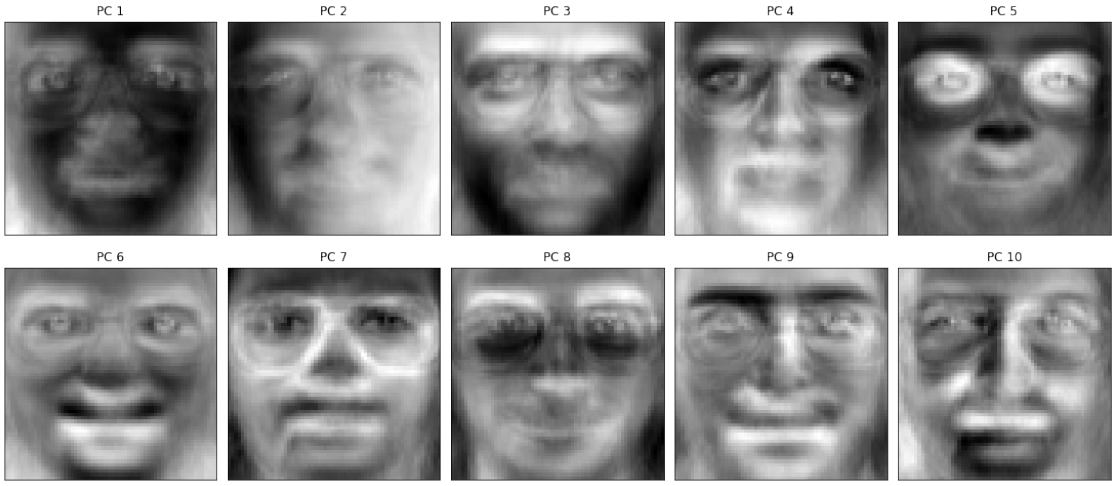
[26]:
```
(400, 4096)
```

Hacemos PCA y vemos sus componentes principales. ¿Cuántos componentes usar?

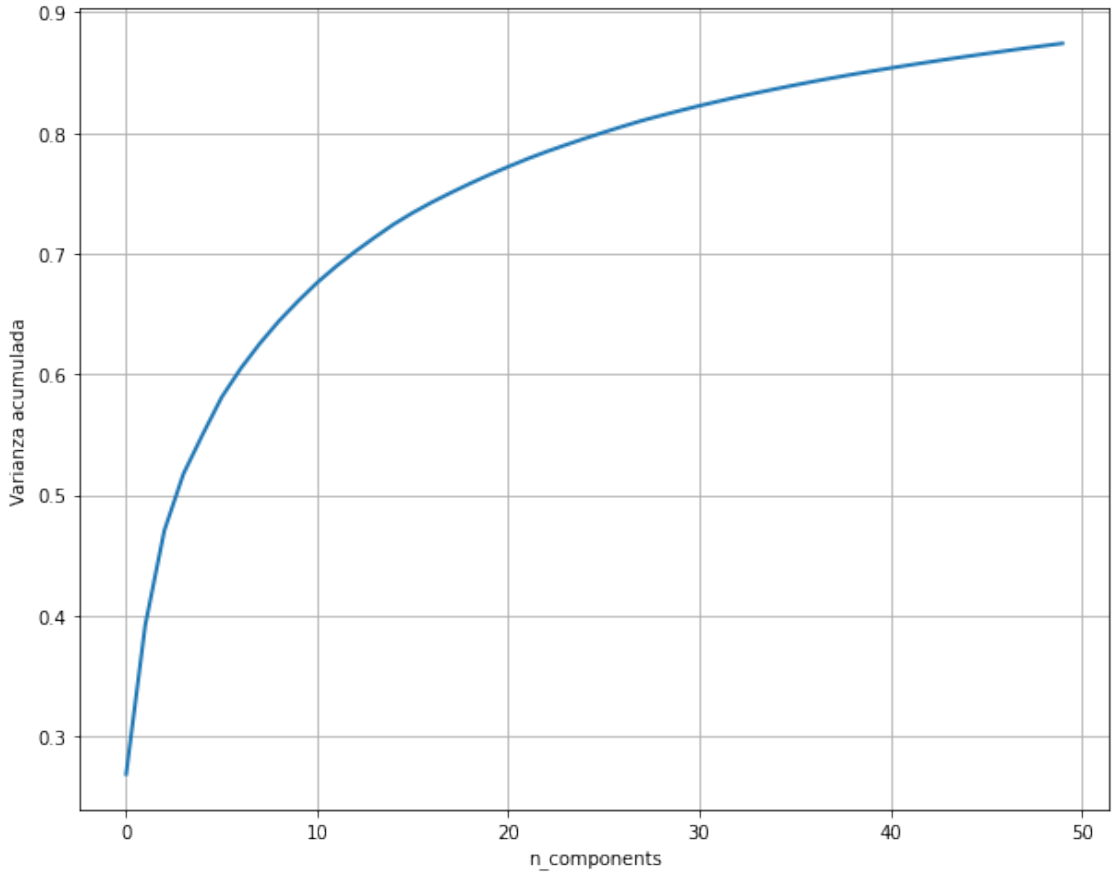
[27]:
```
n_comp = 50 # numero de componentes
faces_pca = PCA(n_comp)
faces_proj = faces_pca.fit_transform(faces_scale)
eigenfaces = faces_pca.components_.reshape((n_comp, 64, 64))
```

[28]:
```
fig = plt.figure(figsize=(15,5))
fig.subplots_adjust(left=0, right=1, bottom=0, top=1.3, hspace=0.05, wspace=0.
↪05)
# Graficamos los primeros 10 componentes
```

```
for i in range(10):
    ax = fig.add_subplot(2, 5, i+1, xticks=[], yticks=[], title = 'PC␣
 ↪'+str(i+1))
    ax.imshow(np.reshape(faces_pca.components_[i,:], (64,64)), cmap=plt.cm.
 ↪gray, interpolation='nearest')
```



[9]:
```
## varianza explicada
plt.figure(figsize=(10, 8))
plt.plot(np.cumsum(faces_pca.explained_variance_ratio_), linewidth=2)
plt.grid(), plt.axis('tight'), plt.xlabel('n_components'), plt.ylabel('Varianza␣
 ↪acumulada')
plt.show()
```



Visualización de los primeros dos eigenfaces. Usamos una visualización interactiva con Bokeh. Asegúrate de tener instalado los componentes del módulo

[35]:
```
from bokeh.plotting import figure, output_file, show, ColumnDataSource
from bokeh.models import HoverTool
#from bokeh.palettes import brewer, Viridis256
import re

# para ordenar los archivos según su numeración...
def sorted_alphanumeric(data):
    convert = lambda text: int(text) if text.isdigit() else text.lower()
    alphanum_key = lambda key: [ convert(c) for c in re.split('([0-9]+)', key) ]
    return sorted(data, key=alphanum_key)

dir_tr = 'data/faces/faces_64/'
sorted_files = sorted_alphanumeric(os.listdir(dir_tr))
name_imgs_tr = [os.path.join(dir_tr,f) for f in sorted_files]
```

[36]:
```
import colorcet as cc
import matplotlib.colors as colors
from colorcet.plotting import swatch, swatches, candy_buttons

# cc.glasbey_bw es un mapa de colores para datos categóricos...
# ver https://colorcet.holoviz.org/user_guide/Categorical.html
lab = np.concatenate([np.repeat(i,10) for i in range(1,41)]) #10 fotos por␣
 ↪sujeto
color_map = [colors.rgb2hex(cc.glasbey_bw[i]) for i in lab-1]

# diccionario con los datos para la grafica de Bokeh
pc_source = dict(x=faces_proj[:,1], y=faces_proj[:,2],
                 label=lab,
                 color=color_map,
                 desc=['sujeto: '+str(i) for i in range(1,faces_proj.
 ↪shape[0]+1)],
                 imgs=name_imgs_tr)
```

[37]:
```
output_file("eigenfaces.html")

source = ColumnDataSource(data = pc_source)

hover = HoverTool(
        tooltips="""
        <div>
            <div>
                <img
                    src="@imgs" height="100" alt="@imgs" width="100"
                    style="float: left; margin: 0px 15px 15px 0px;"
                    border="2"
                ></img>
            </div>
            <div>
                <span style="font-size: 17px; font-weight: bold;">@desc</span>
                <span style="font-size: 15px; color: #966;">[$index]</span>
            </div>
            <div>
                <span style="font-size: 15px;">Location</span>
                <span style="font-size: 10px; color: #696;">($x, $y)</span>
            </div>
        </div>
        """
    )

p = figure(plot_width=2000, plot_height=800, tools=[hover], title="Eigenfaces",
          x_axis_label="PC 1",y_axis_label="PC 2")
```

```
p.circle('x', 'y', size=5,  color='color', source=source)

show(p)
```