



Instituto Tecnológico y de Estudios Superiores de Monterrey

Maestría en Inteligencia Artificial Aplicada – MNA

Materia: Ciencia y Analítica de Datos.

Profesora: María de la Paz

Tarea semana 6:

Reducción de Dimensiones

Equipo 24:

-Rafael José Mateo Comprés A01793054

Haz doble clic (o ingresa) para editar

Bienvenido al notebook

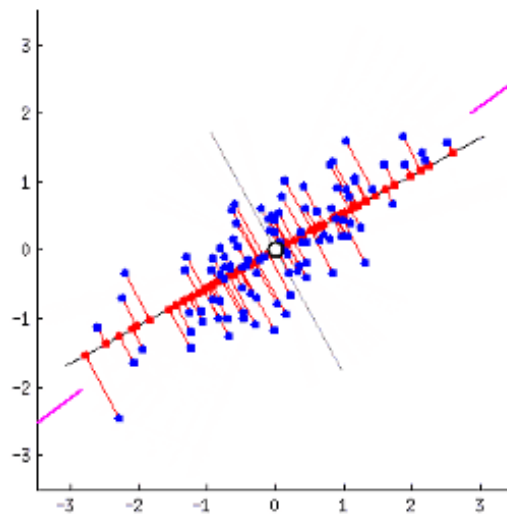
Repaso de Reducción de dimensiones

El objetivo es que entendamos de una manera visual, que es lo que pasa cuando nosotros seleccionamos cierto número de componentes principales o % de variabilidad de una base de datos.

Primero entenderemos, que pasa adentro de PCA que se basa en lo siguiente a grandes razgos:

Análisis de Componentes Principales

El análisis de datos multivariados involucra determinar transformaciones lineales que ayuden a entender las relaciones entre las características importantes de los datos. La idea central del Análisis de Componentes Principales (PCA) es reducir las dimensiones de un conjunto de datos que presenta variaciones correlacionadas, reteniendo una buena proporción de la variación presente en dicho conjunto. Esto se logra obteniendo la transformación a un nuevo conjunto de variables: los componentes principales (PC). Cada PC es una combinación lineal con máxima varianza en dirección ortogonal a los demás PC.



Para entender un poco más de PCA y SVD, visita el siguiente link: *Truco: Prueba entrar con tu*

cuenta del tec :)

<https://towardsdatascience.com/pca-and-svd-explained-with-numpy-5d13b0d2a4d8>

Basicamente, vamos a seguir los siguientes pasos:

1. Obtener la covarianza. OJO: X tiene sus datos centrados :)

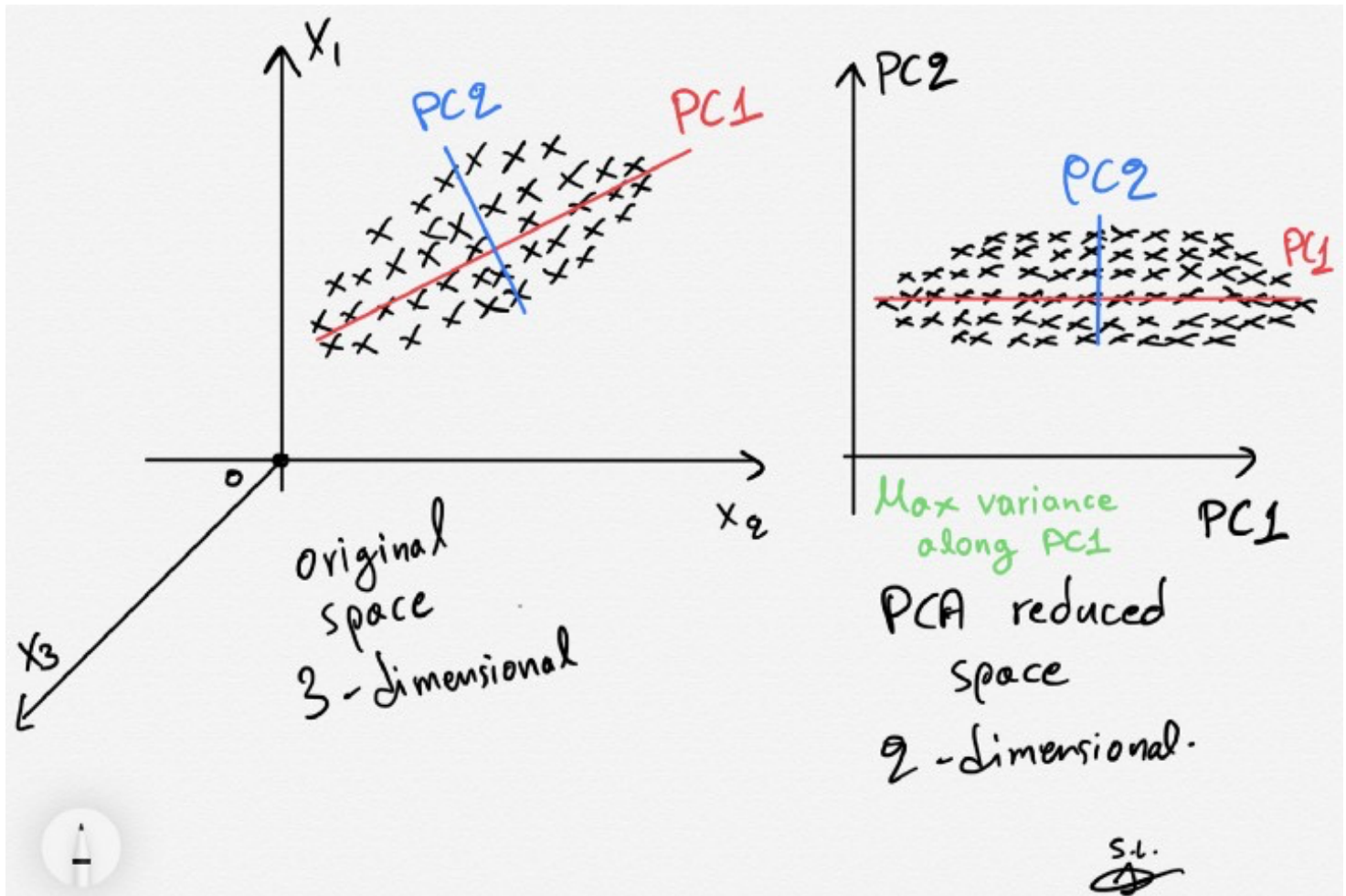
$$\mathbf{C} = \frac{\mathbf{X}^T \mathbf{X}}{n - 1}$$

2. Los componentes principales se van a obtener de la eigen descomposicion de la matriz de covarianza.

$$\mathbf{C} = \mathbf{W} \mathbf{\Lambda} \mathbf{W}^{-1}$$

3. Para la reducción de dimensiones vamos a seleccionar k vectores de W y proyectaremos nuestros datos.

$$\mathbf{X}_k = \mathbf{X} \mathbf{W}_k$$



Ejercicio 1, Descomposición y composición

Descomposición

Encuentra los eigenvalores y eigenvectores de las siguientes matrices

$$A = \begin{pmatrix} 3, 0, 2 \\ 3, 0, -2 \\ 0, 1, 1 \end{pmatrix} \quad A2 = \begin{pmatrix} 1, 3, 8 \\ 2, 0, 0 \\ 0, 0, 1 \end{pmatrix} \quad A3 = \begin{pmatrix} 5, 4, 0 \\ 1, 0, 1 \\ 10, 7, 1 \end{pmatrix}$$

y reconstruye la matriz original a través de las matrices $W D W^{-1}$ (OJO. Esto es lo mismo de la ecuación del paso 2 solo le cambiamos la variable a la matriz diagonal)

▼ Eigenvalores y eigenvectores

```

###-----EJEMPLO DE EIGENVALORES
import numpy as np
from numpy import array
from numpy.linalg import eig
# define la matriz
A = array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print("-----Matriz original-----")
print(A)
print("-----")
# calcula la eigendescomposición
values, vectors = eig(A)
print(values) #D
print(vectors) #W

#Ejemplo de reconstrucción

values, vectors = np.linalg.eig(A)

W = vectors
Winv = np.linalg.inv(W)
D = np.diag(values)
#la matriz B tiene que dar igual a A
#reconstruye la matriz
print("-----Matriz reconstruida-----")
# Realiza la reconstruccion de B=W*D*Winv, te da lo mismo de A?
#ojo, estas multiplicando matrices, no escalares ;)
#TU CODIGO AQUÍ-----
B= W.dot(D).dot(Winv)
print(B)
print("-----")

```

```

-----Matriz original-----
[[1 2 3]
 [4 5 6]
 [7 8 9]]
-----
[ 1.61168440e+01 -1.11684397e+00 -1.30367773e-15]
[[-0.23197069 -0.78583024  0.40824829]
 [-0.52532209 -0.08675134 -0.81649658]
 [-0.8186735   0.61232756  0.40824829]]
-----Matriz reconstruida-----
[[1.  2.  3.]
 [4.  5.  6.]
 [7.  8.  9.]]
-----

```

```

# Se define la matriz original
A = array([[5,8,2], [4,5,6], [3,5,8]])
print("-----Matriz original-----")
print(A)
print("-----")

```

```

#Se obtienen los eigenvalores y sus correspondientes eigenvectores
values, vectors = eig(A)
Winv = np.linalg.inv(vectors)
D = np.diag(values)

```

```

#Se hace la reconstrucción a la matriz original
print("-----Matriz reconstruida-----")
B= vectors.dot(D).dot(Winv)
print(B)
print("-----")

```

```

-----Matriz original-----
[[5 8 2]
 [4 5 6]
 [3 5 8]]
-----
-----Matriz reconstruida-----
[[5.  8.  2.]
 [4.  5.  6.]
 [3.  5.  8.]]
-----

```

```
#Matriz 2
```

```
# Se define la matriz original
A = array([[11,4,9], [2,6,9], [10,12, 25]])
print("-----Matriz original-----")
print(A)
print("-----")
```

```
#Descomposición
```

```
#Se obtienen los eigenvalores y sus correspondientes eigenvectores
values, vectors = eig(A)
Winv = np.linalg.inv(vectors)
D = np.diag(values)
```

```
#Se hace la reconstrucción a la matriz original
print("-----Matriz reconstruida-----")
B= vectors.dot(D).dot(Winv)
print(B)
print("-----")
```

```
-----Matriz original-----
[[11  4  9]
 [ 2  6  9]
 [10 12 25]]
-----
```

```
-----Matriz reconstruida-----
[[11.  4.  9.]
 [ 2.  6.  9.]
 [10. 12. 25.]]
-----
```

```
#Matriz 3
```

```
# Se define la matriz original
A = array([[15,68,12], [44,65,36], [34,35,81]])
print("-----Matriz original-----")
print(A)
print("-----")
```

```
#Descomposición
```

```
#Se obtienen los eigenvalores y sus correspondientes eigenvectores
values, vectors = eig(A)
Winv = np.linalg.inv(vectors)
D = np.diag(values)
```

```
#Se hace la reconstrucción a la matriz original
print("-----Matriz reconstruida-----")
B= vectors.dot(D).dot(Winv)
print(B)
print("-----")
```

```
-----Matriz original-----
[[15 68 12]
 [44 65 36]
 [34 35 81]]
-----
```

```
-----Matriz reconstruida-----
[[15. 68. 12.]
 [44. 65. 36.]
 [34. 35. 81.]]
-----
```


¿Qué significa reducir dimensiones?

Esto será cuando proyectemos a ese espacio de los componentes principales pero no los seleccionemos todos, solo los más importantes y viajemos de regreso a nuestras unidades a través de una proyección.

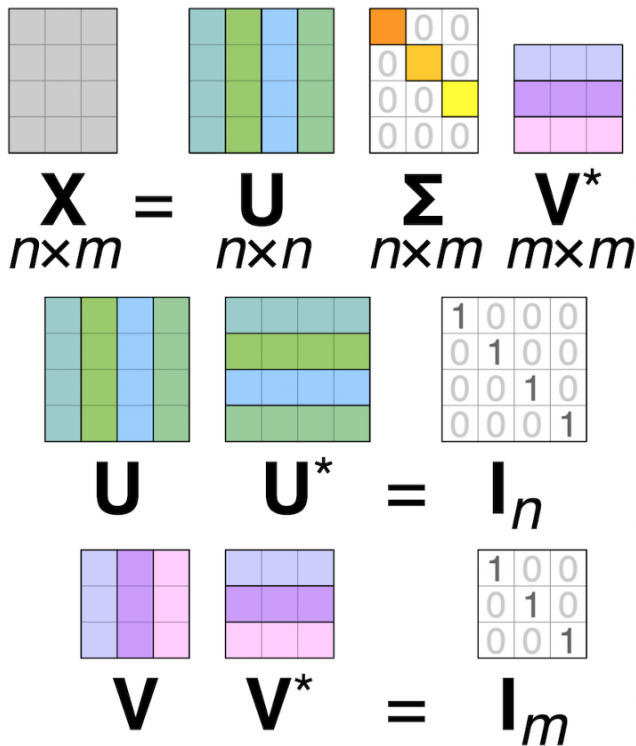
Es decir: Unidades-PC PC-Unidades

Veamoslo gráficamente, ¿qué pasa con esa selección de los PCs y su efecto?.

Para ello usaremos Singular Value Descomposition (SVD).

Singular Value Descomposition(SVD)

Es otra descomposición que tambien nos ayudara a reducir dimensiones.



$$\begin{matrix}
 \begin{matrix} \text{4x4 grid} \end{matrix} & = & \begin{matrix} \text{4x4 grid} \end{matrix} & \begin{matrix} \text{4x4 grid} \end{matrix} & \begin{matrix} \text{4x4 grid} \end{matrix} \\
 \mathbf{X} & = & \mathbf{U} & \mathbf{\Sigma} & \mathbf{V}^* \\
 n \times m & & n \times n & n \times m & m \times m
 \end{matrix}$$

$$\begin{matrix}
 \begin{matrix} \text{4x4 grid} \end{matrix} & \begin{matrix} \text{4x4 grid} \end{matrix} & = & \begin{matrix} \text{4x4 grid} \end{matrix} \\
 \mathbf{U} & \mathbf{U}^* & = & \mathbf{I}_n
 \end{matrix}$$

$$\begin{matrix}
 \begin{matrix} \text{4x4 grid} \end{matrix} & \begin{matrix} \text{4x4 grid} \end{matrix} & = & \begin{matrix} \text{4x4 grid} \end{matrix} \\
 \mathbf{V} & \mathbf{V}^* & = & \mathbf{I}_m
 \end{matrix}$$

▼ Ejercicio 2

Juega con Lucy, una cisne, ayudala a encontrar cuantos valores singulares necesita para no perder calidad a través de SVD. Posteriormente usa 3 imágenes de tu preferencia y realiza la misma acción :D

A esto se le llama **compresión de imagenes** :o

Haz doble clic (o ingresa) para editar

```
from six.moves import urllib
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np

plt.style.use('classic')
img = Image.open(urllib.request.urlopen('https://biblioteca.acropolis.org/wp-conter
#img = Image.open('lucy.jpg')
imggray = img.convert('LA')
imgmat = np.array(list(imggray.getdata(band=0)),float)

print(imgmat)

imgmat.shape = (imggray.size[1],imggray.size[0])

plt.figure(figsize=(9,6))
plt.imshow(imgmat,cmap='gray')
plt.show()
print(img)
```

```
[ 72.  73.  74. ... 48. 47. 47.]
```



```
<PIL.Image.Image image mode=LA size=1024x660 at 0x7F4DC7543A10>
```

```
U,D,V = np.linalg.svd(imgmat)  
imgmat.shape
```

```
(660, 1024)
```

```
U.shape
```

```
(660, 660)
```

```
V.shape
```

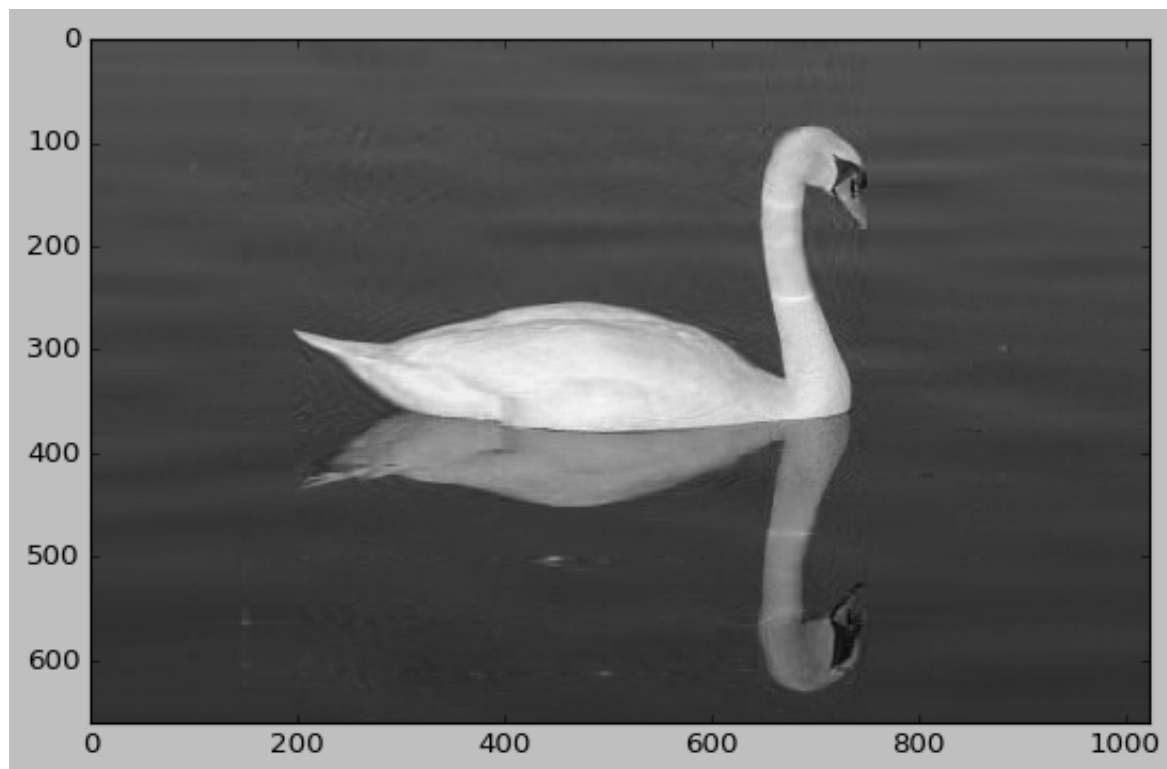
```
(1024, 1024)
```

```

#Cuantos valores crees que son necesarios?
#A=U*D*V
#aqui los elegiremos-----
# por las dimensiones de este caso en particular
#iremos de 0-660, siendo 660 como normalmente están los datos
#con 50 podemos observar que Lucy se ve casi igual, es decir conservamos aquello q
# realidad estaba aportando a la imagen en este caso :D por medio de la variabilidad
#juega con el valor nvalue y ve que pasa con otros valores
nvalue = 45
#-----
reconstimg = np.matrix(U[:, :nvalue])*np.diag(D[:nvalue])*np.matrix(V[:nvalue, :])
#ve las dimensiones de la imagen y su descomposicion
#660x1024= U(660X660)D(660X1024)V(1024x1024)
#=#U(660Xnvalues)D(nvaluesXnvalue)V(nvaluesx1024)

#=#U(660X50)(50X50)(50X1024)
plt.imshow(reconstimg, cmap='gray')
plt.show()
print("Felicidades la imagen está comprimida")

```



Felicidades la imagen está comprimida

¡Ahora es tu turno!, comprime 3 imagenes

```
#imagen 1

plt.style.use('classic')

#Importamos la imagen
img = Image.open(urllib.request.urlopen('https://hips.hearstapps.com/ghk.h-cdn.co/a

#Convertimos la imagen a escala de grises

imggray = img.convert('LA')

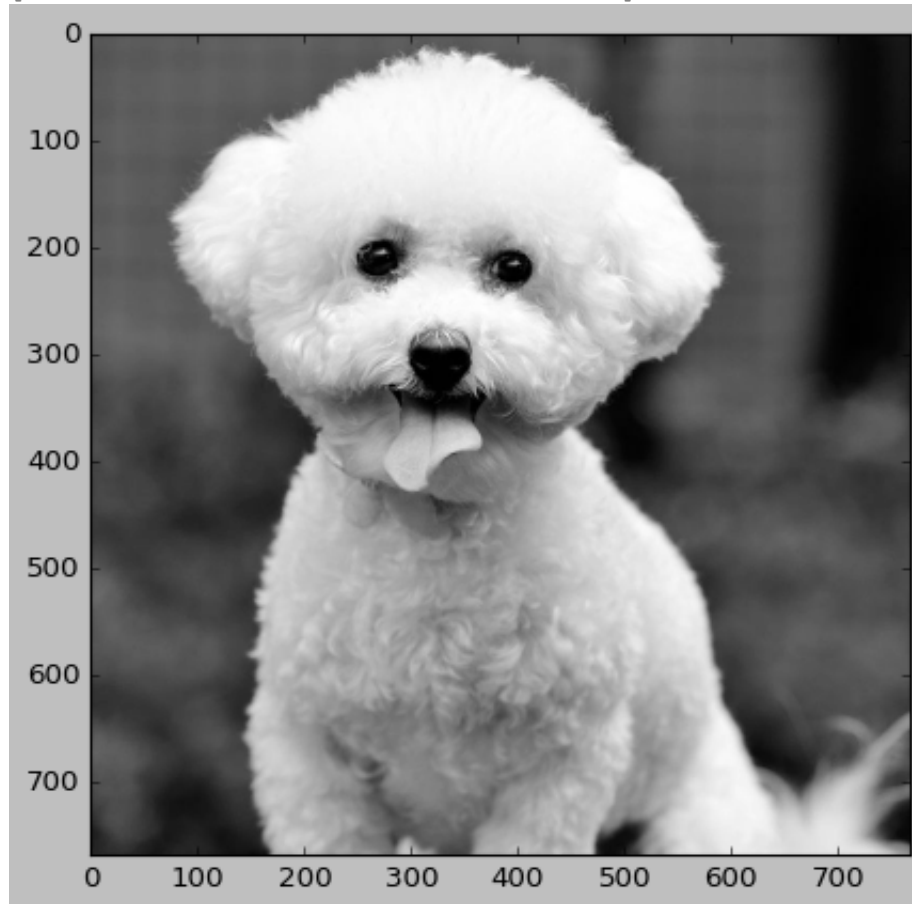
#Obtenemos los pixeles de la imagen y lo convertimos a un np array
imgmat = np.array(list(imggray.getdata(band=0)),float)

print(imgmat)

#Definimos el shape del array de los pixeles
imgmat.shape = (imggray.size[1],imggray.size[0])

#Mostramos la imagen
plt.figure(figsize=(9,6))
plt.imshow(imgmat,cmap='gray')
plt.show()
print(img)
```

```
[ 88.  88.  88. ... 154. 152. 150.]
```



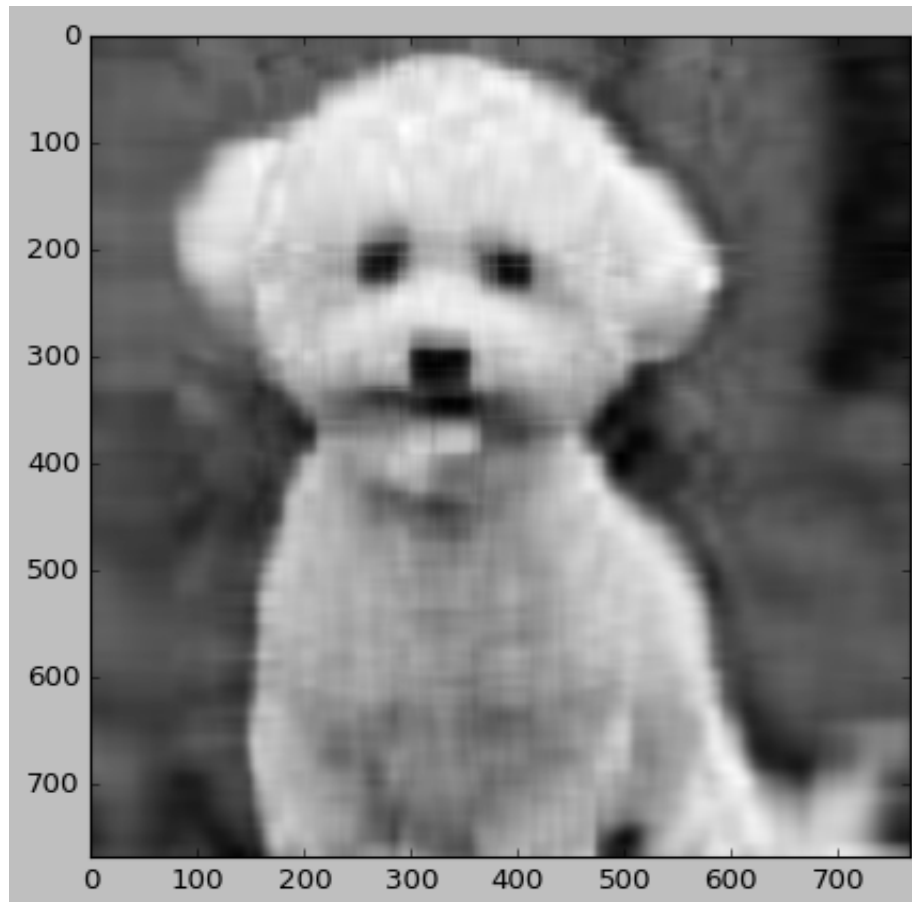
```
<PIL.Image.Image image mode=LA size=768x768 at 0x7F4DBF2711D0>
```

```
#Descomponemos la matriz de pixeles en tres matrices  
U,D,V = np.linalg.svd(imgmat)
```

```
nvalue = 15
```

```
#Reconstruimos la imagen aplicando el nvalue  
reconstimg = np.matrix(U[:, :nvalue])*np.diag(D[:nvalue])*np.matrix(V[:nvalue, :])
```

```
plt.imshow(reconstimg, cmap='gray')  
plt.show()
```




```
#imagen 2

plt.style.use('classic')

#Importamos la imagen
img = Image.open(urllib.request.urlopen('https://icatcare.org/app/uploads/2018/07/1
#Convertimos la imagen a escala de grises

imggray = img.convert('LA')

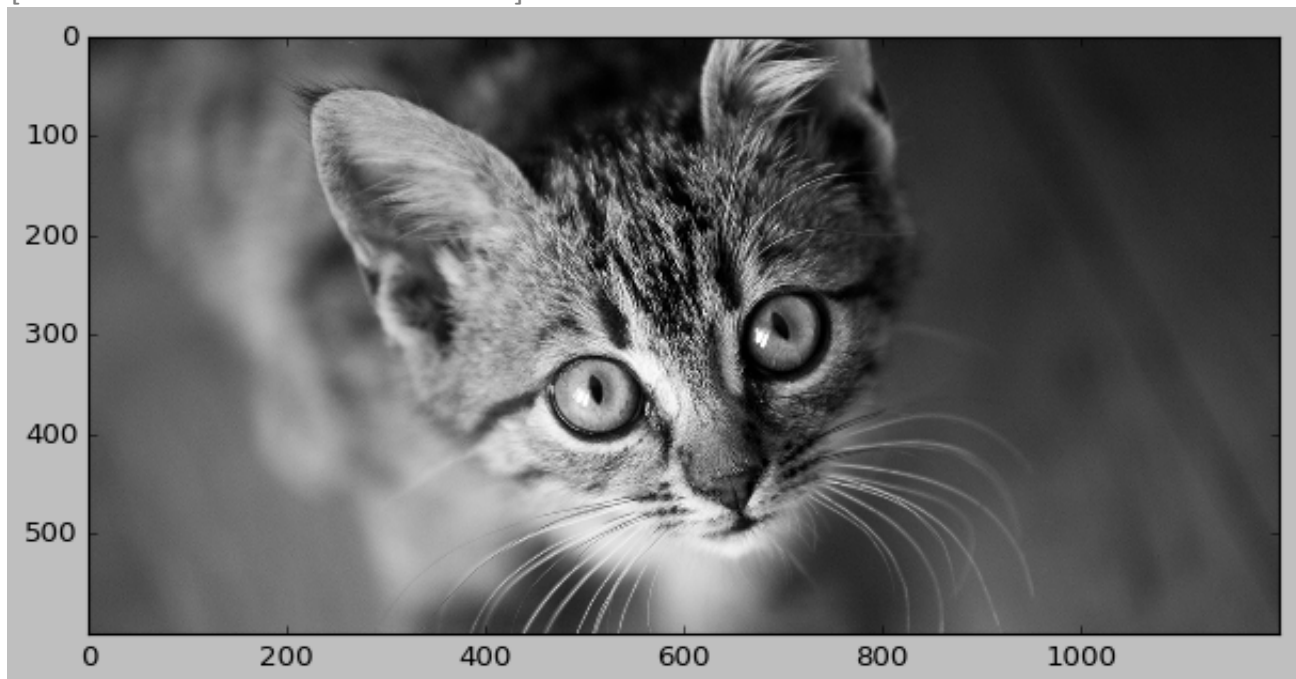
#Obtenemos los pixeles de la imagen y lo convertimos a un np array
imgmat = np.array(list(imggray.getdata(band=0)),float)

print(imgmat)

#Definimos el shape del array de los pixeles
imgmat.shape = (imggray.size[1],imggray.size[0])

#Mostramos la imagen
plt.figure(figsize=(9,6))
plt.imshow(imgmat,cmap='gray')
plt.show()
print(img)
```

```
[48. 37. 33. ... 70. 67. 73.]
```



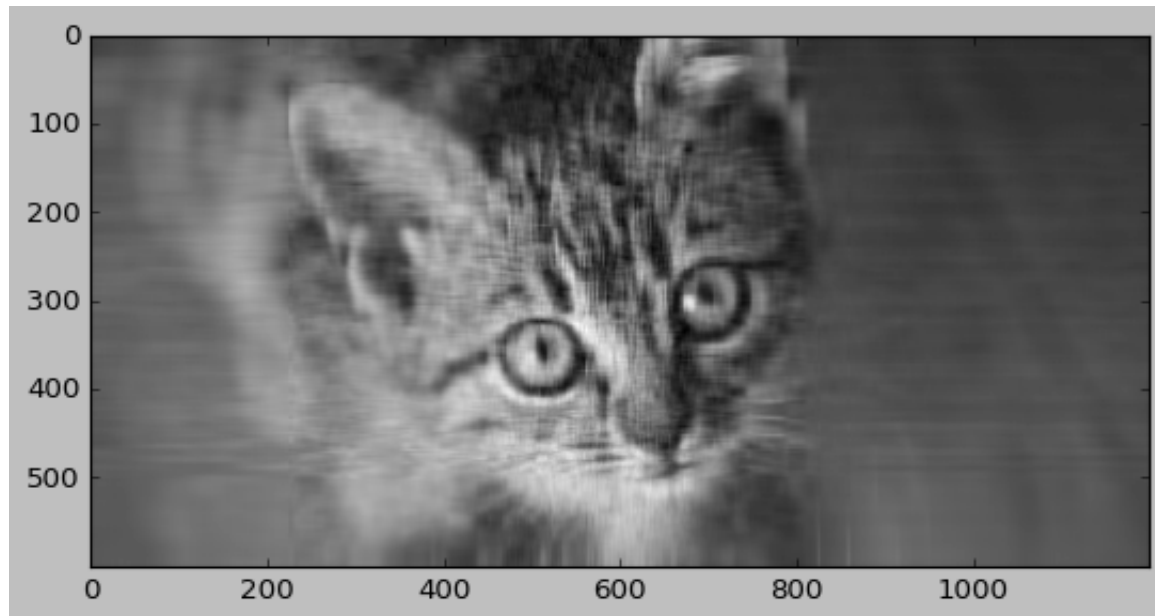
```
<PIL.Image.Image image mode=LA size=1200x600 at 0x7F4DBF40B290>
```

```
#Descomponemos la matriz de pixeles en tres matrices
U,D,V = np.linalg.svd(imgmat)

nvalue = 20

#Reconstruimos la imagen aplicando el nvalue
reconstimg = np.matrix(U[:, :nvalue])*np.diag(D[:nvalue])*np.matrix(V[:nvalue, :])

plt.imshow(reconstimg, cmap='gray')
plt.show()
print("Felicidades la imagen está comprimida")
```



Felicidades la imagen está comprimida

```
#imagen 3

plt.style.use('classic')
img = Image.open(urllib.request.urlopen('https://www.animalife.co.uk/wp-content/upl
#Convertimos la imagen a escala de grises

imggray = img.convert('LA')

#Obtenemos los pixeles de la imagen y lo convertimos a un np array
imgmat = np.array(list(imggray.getdata(band=0)),float)

print(imgmat)
```

```
#Definimos el shape del array de los pixeles  
imgmat.shape = (imggray.size[1],imggray.size[0])
```

```
#Mostramos la imagen  
plt.figure(figsize=(9,6))  
plt.imshow(imgmat,cmap='gray')  
plt.show()  
print(img)
```

```
[248. 248. 248. ... 197. 197. 198.]
```



```
<PIL.Image.Image image mode=LA size=1100x733 at 0x7F4DB50F8E90>
```

```
#Descomponemos la matriz de los pixeles en tres matrices
U,D,V = np.linalg.svd(imgmat)

nvalue = 50

#Reconstruimos la imagen con el valor nvalue definido arriba
reconstimg = np.matrix(U[:, :nvalue])*np.diag(D[:nvalue])*np.matrix(V[:nvalue, :])

plt.imshow(reconstimg, cmap='gray')
plt.show()
print("Felicidades la imagen está comprimida")
```



Felicidades la imagen está comprimida

▼ Ejercicio 3

Feature importances

Para este ejercicio, te pediremos que sigas el tutorial de la siguiente pagina:

<https://towardsdatascience.com/pca-clearly-explained-how-when-why-to-use-it-and-feature-importance-a-guide-in-python-7c274582c37e>

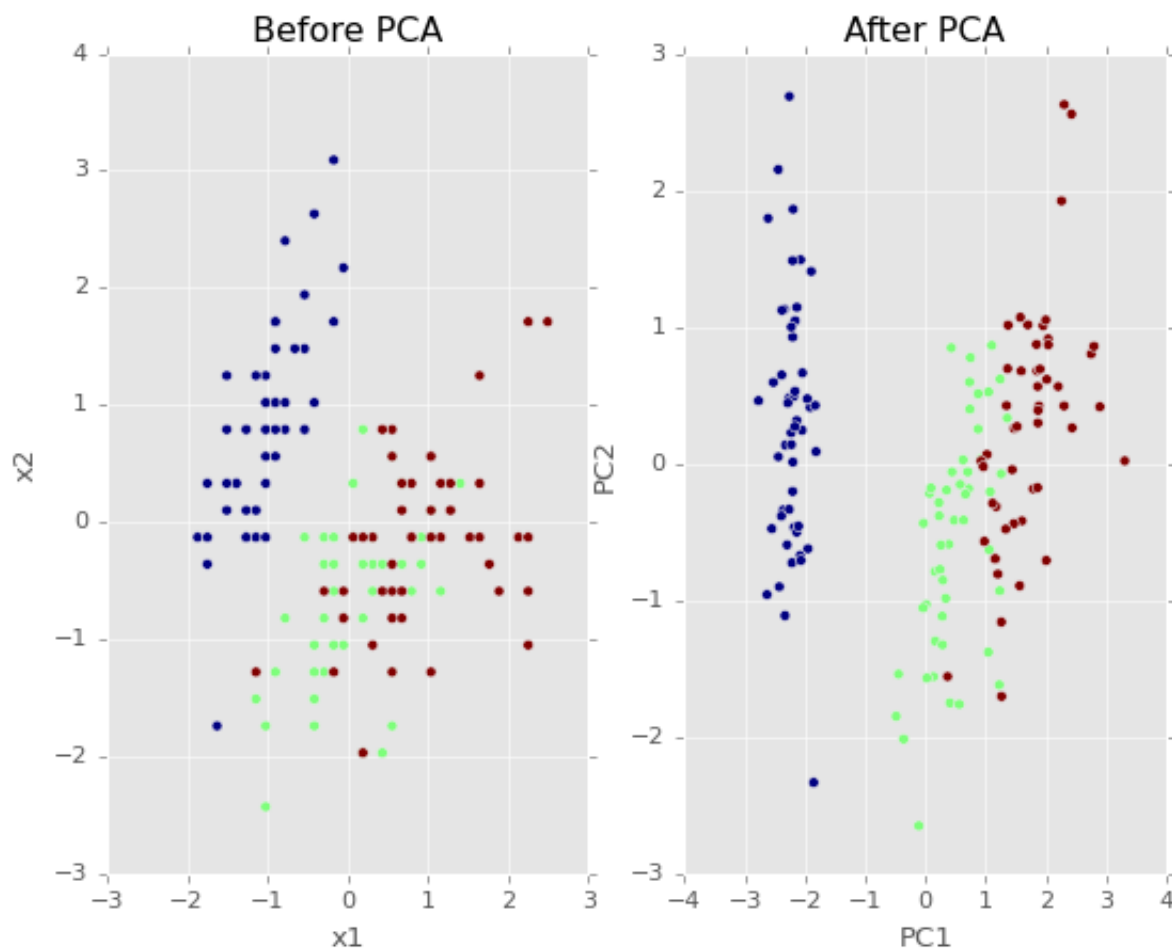
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA
import pandas as pd
from sklearn.preprocessing import StandardScaler
plt.style.use('ggplot')

# Cargamos el dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Escalamos los datos, centrando la media a cero e igualando las varianzas a 1
scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)
# Obtenemos los PCA
pca = PCA(n_components=2) # Solo 2 componentes
X_new = pca.fit_transform(X) # Obtén la transformación
```

```
#Realizamos un gráfico de las variables originales y luego otro de los componentes
fig, axes = plt.subplots(1,2)
axes[0].scatter(X[:,0], X[:,1], c=y)
axes[0].set_xlabel('x1')
axes[0].set_ylabel('x2')

axes[0].set_title('Before PCA')
axes[1].scatter(X_new[:,0], X_new[:,1], c=y)
axes[1].set_xlabel('PC1')
axes[1].set_ylabel('PC2')
axes[1].set_title('After PCA')
plt.show()
```



Describe lo relevante del ejercicio y que descubriste de las variables analizadas.

```
#Imprimimos la varianza explicada. Se observa que el modelo explica alrededor del 7
print(pca.explained_variance_ratio_)
```

```
[0.72962445 0.22850762]
```

```
#Calculamos las covarianzas
np.cov(X_new.T)
```

```
array([[2.93808505e+00, 5.33928780e-16],
       [5.33928780e-16, 9.20164904e-01]])
```

```
pca.explained_variance_
```

```
array([2.93808505, 0.9201649 ])
```

```
#Imprimimos el valor de los componentes principales
print(abs( pca.components_ ))
```

```
[[0.52106591 0.26934744 0.5804131  0.56485654]
 [0.37741762 0.92329566 0.02449161 0.06694199]]
```

```
#Definimos una función para hacer un biplot y ver el comportamiento del modelo de n
def biplot(score, coeff , y):
    ...

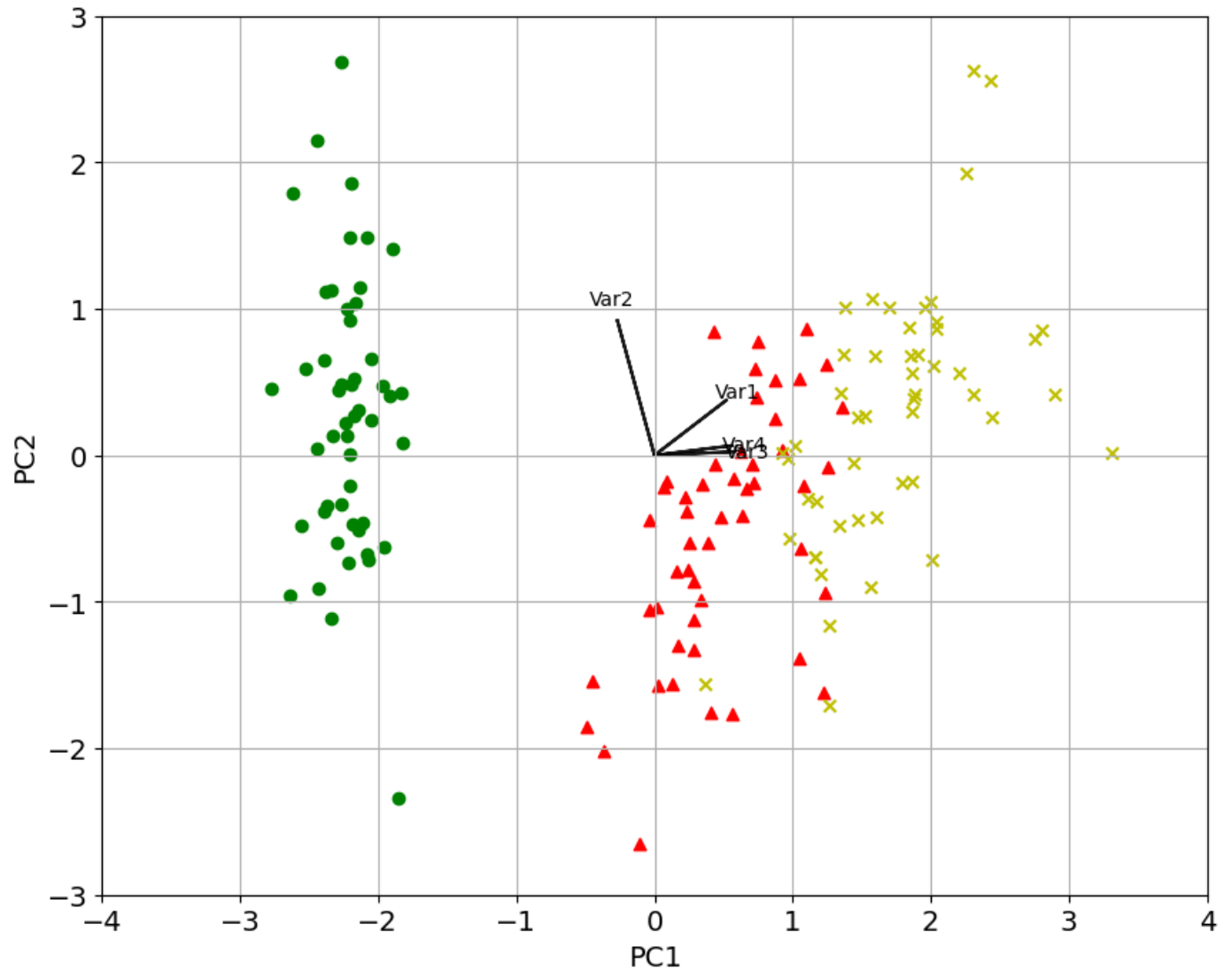
    Author: Serafeim Loukas, serafeim.loukas@epfl.ch
    Inputs:
        score: the projected data
        coeff: the eigenvectors (PCs)
        y: the class labels
    ...

    xs = score[:,0] # Proyección en PC1
    ys = score[:,1] # Proyección en PC2
    n = coeff.shape[0] # Número de variables
    plt.figure(figsize=(10,8), dpi=100)
    classes = np.unique(y)
    colors = ['g','r','y']
    markers=['o','^','x']
    for s,l in enumerate(classes):
        plt.scatter(xs[y==l],ys[y==l], c = colors[s], marker=markers[s]) # Damos co
    for i in range(n):
        #Graficamos como flecha los scores de cada variable
        plt.arrow(0, 0, coeff[i,0], coeff[i,1], color = 'k', alpha = 0.9,linestyle
        plt.text(coeff[i,0]* 1.15, coeff[i,1] * 1.15, "Var"+str(i+1), color = 'k',

#Definimos los labels de cada componente y el límite de los ejes
plt.xlabel("PC{}".format(1), size=14)
plt.ylabel("PC{}".format(2), size=14)
limx= int(xs.max()) + 1
limy= int(ys.max()) + 1
plt.xlim([-limx,limx])
plt.ylim([-limy,limy])
plt.grid()
plt.tick_params(axis='both', which='both', labelsize=14)
```



```
import matplotlib as mpl
mpl.rcParams.update(mpl.rcParamsDefault)
# Llamamos la función para plotear los componentes
biplot(X_new[:,0:2], np.transpose(pca.components_[0:2, :]), y)
plt.show()
```



```
# Se evidencia que las variables 3 y 4 están altamente correlacionadas
np.corrcoef(X[:,2], X[:,3])[1,0]
```

```
0.9628654314027957
```

```
# Las variables 2 y 3 están correlacionadas de manera negativa
np.corrcoef(X[:,1], X[:,2])[1,0]
```

```
-0.42844010433054014
```

¿Qué es feature importance y para que nos sirve?

R: El feature importance nos permite identificar las variables que más impactan en los componentes calculados en la técnica PCA. Esto se hace identificando aquellas variables donde sus coeficientes tengan las magnitudes más altas.

En otras palabras, esto significa que tanto contribuye un feature en la variabilidad explicada por el componente.

Con relación a su uso, esto nos podría servir para reducir las dimensiones de un conjunto de datos con muchas variables y a la vez entender mejor la composición de cada componente.

¿Qué hallazgos fueron los más relevantes durante el análisis del ejercicio?

R: Durante el ejercicio se pudo observar que 2 componentes explican cerca del 70% de la variación. También se evidenció que las variables 1,3,4 son las que más impacto tienen en el componente 1, mientras que las variables 1 y 2 son las que más contribuyen en el componente 2.

Por último, fue posible observar una alta correlación positiva entre las variables 3 y 4 y una moderada correlación negativa entre las variables 2 y 3. Esto fue posible verlo desde el biplot, donde se observa claramente la dirección de los vectores de estas variables.

¿Dónde lo aplicarías o te sería de utilidad este conocimiento?

R: Esto sería útil en la reducción de dimensiones cuando se cuenta con un modelo con muchas variables. Por ejemplo, esto nos permitiría reducir las dimensiones a unos cuantos componentes que explique un alto porcentaje de la variabilidad del conjunto de datos.

[Productos pagados de Colab](#) - [Cancela los contratos aquí](#)

