Haz doble clic (o pulsa Intro) para editar

Actividad Semanal -- 5 Repaso Transformación y reducción de dimensiones

Hector Manuel Gonzalez Villarreal a00178679

Prof: Maria de la Paz Rico

27 de Oct del 2022

Bienvenido al notebook

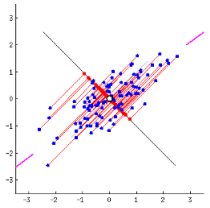
Repaso de Reducción de dimensiones

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 :)

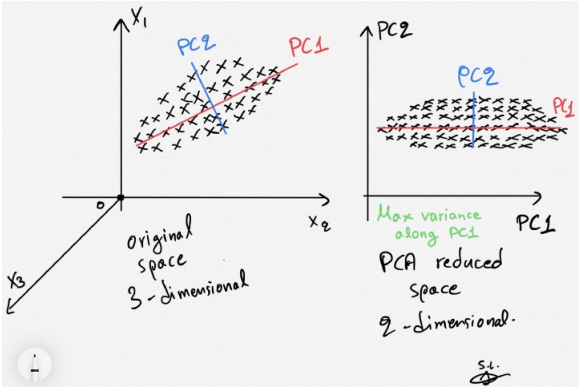
$$\mathbf{C} = \frac{\mathbf{X}^\top \mathbf{X}}{n-1}$$

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

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

y reconstruye la matriz original a traves de las matrices WDW^{-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 AQUI-------------------------------------
B= np.dot(np.dot(W,D),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.]]
------------------------------
```

```
A = array([[3, 0, 2], [3, 0, -2], [0, 1, 1]])
values, vectors = eig(A)
print(values) #D
print(vectors) #W
```

```
[3.54451153+0.j         0.22774424+1.82582815j 0.22774424-1.82582815j]
[[-0.80217543+0.j         -0.04746658+0.2575443j  -0.04746658-0.2575443j ]
 [-0.55571339+0.j          0.86167879+0.j          0.86167879-0.j        ]
 [-0.21839689+0.j         -0.16932106-0.40032224j -0.16932106+0.40032224j]]
```

```
#Matriz 1
A2 = array([[1, 3, 8], [2, 0, 0], [0, 0, 1]])
values, vectors = eig(A2)
print(values) #D
print(vectors) #W
```

```
[ 3. -2.  1.]
[[ 0.83205029 -0.70710678 -0.42399915]
 [ 0.5547002   0.70710678 -0.8479983 ]
 [ 0.          0.          0.31799936]]
```

```
#Matriz 2
A3 = array([[5, 4, 0], [1, 0, 1], [10, 7, 1]])
values, vectors = eig(A3)
print(values) #D
print(vectors) #W
```

```
[ 6.89167094 -0.214175   -0.67749594]
[[ 0.3975395   0.55738222  0.57580768]
 [ 0.18800348 -0.72657211 -0.81728644]
 [ 0.89811861 -0.40176864 -0.02209943]]
```

```
#Matriz 3
W = vectors
Winv = np.linalg.inv(W)
B2= np.dot(np.dot(W,A3),Winv)
print(B2)
```

```
[[ 30.40328343  20.26227621  -8.45403693]
 [-26.2765603  -16.79369679   6.28405288]
 [ 19.93596004  14.76999239  -7.60958664]]
```

¿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.

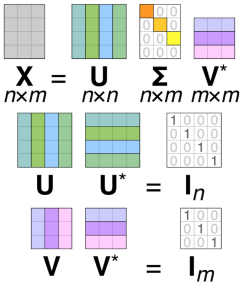
Es decir: Unidades-PC PC-Unidades

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.



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

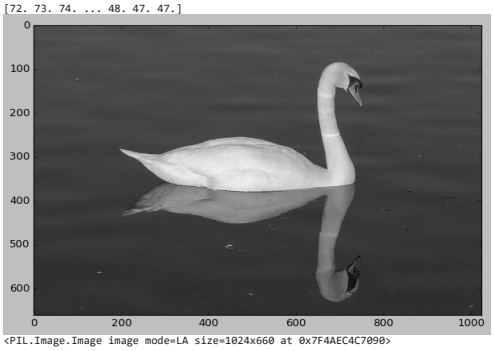
A esto se le llama **compresión de imagenes** :o

Haz doble clic (o pulsa Intro) 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-content/uploads/2015/03/Cisne.jpg')).convert('LA')
#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 0x7F4AEC4C7090>
```

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

```
(660, 1024)
```

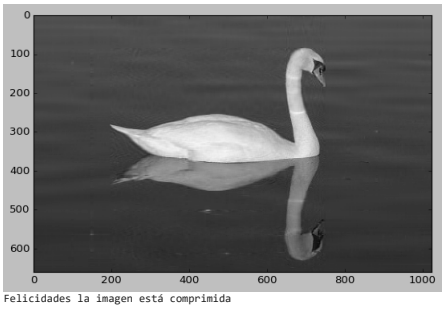
```
U.shape
```

```
(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 que en
# 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 = 50
#------------------------------
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")
```



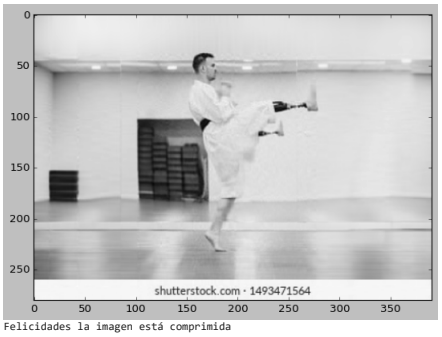

```
Felicidades la imagen está comprimida
```

¡Ahora es tu turno!, comprime 3 imagenes

```
#imagen 1
plt.style.use('classic')
img = Image.open(urllib.request.urlopen('https://image.shutterstock.com/image-photo/man-prosthetic-leg-practicing-karate-260nw-1493471564.jpg')).convert('LA')
#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)
```

```
[199. 201. 203. ... 255. 255. 255.]
```

```
U,D,V = np.linalg.svd(imgmat)
nvalue = 50
#------------------------------
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("Imagen 2 comprimida")
```



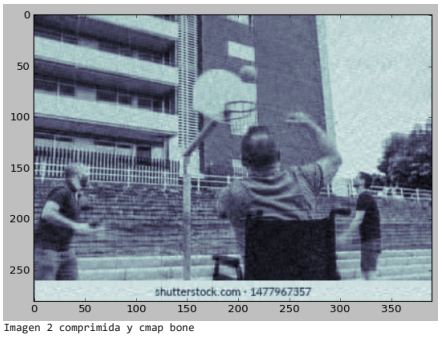

```
Felicidades la imagen está comprimida
```

```
#imagen 2
plt.style.use('classic')
img = Image.open(urllib.request.urlopen('https://image.shutterstock.com/image-photo/joyful-disabled-man-wheelchair-throwing-260nw-1477967357.jpg')).convert('LA')
#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='bone')
plt.show()
print(img)
```

```
[ 49.  52.  55. ... 255. 255. 255.]
```


```
<PIL.Image.Image image mode=LA size=390x280 at 0x7F4AEBF6A7D0>
```

```
U,D,V = np.linalg.svd(imgmat)
nvalue = 50
#------------------------------
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='bone')
plt.show()
print("Imagen 2 comprimida y cmap bone")
```



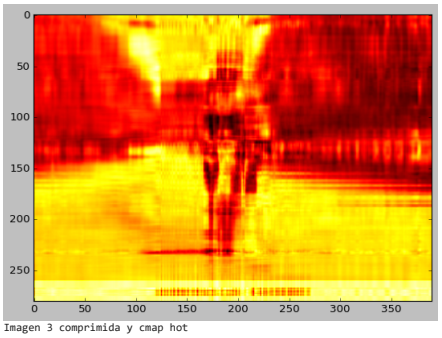

```
Imagen 2 comprimida y cmap bone
```

```
#imagen 3
plt.style.use('classic')
img = Image.open(urllib.request.urlopen('https://image.shutterstock.com/image-photo/athlete-disabilities-amputee-training-cycling-260nw-1459658057.jpg')).convert('LA')
#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='hot')
plt.show()
print(img)
```

```
[ 89.  85.  78. ... 255. 255. 255.]
```


```
<PIL.Image.Image image mode=LA size=390x280 at 0x7F4AEAE9C510>
```

```
U,D,V = np.linalg.svd(imgmat)
nvalue = 10
#------------------------------
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='hot')
plt.show()
print("Imagen 3 comprimida a 10 y cmap hot")
```



```
Imagen 3 comprimida y cmap hot
```

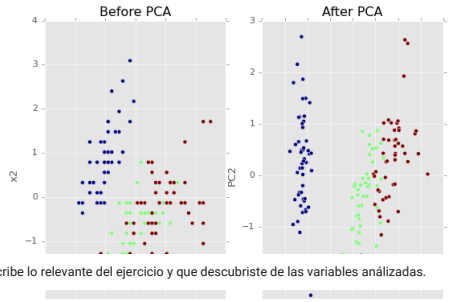
Ejercicio 3

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

```
#tu codigo aqui
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')
# Load the data
iris = datasets.load_iris()
X = iris.data
y = iris.target
# Z-score the features
scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)
# The PCA model
pca = PCA(n_components=2) # estimate only 2 PCs
X_new = pca.fit_transform(X) # project the original data into the PCA space


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 análizadas.

```
print(pca.explained_variance_ratio_)
# array([0.72962445, 0.22850762])
```

```
     [0.72962445 0.22850762]
```

```
np.cov(X_new.T)
array([[2.93808505e+00, 4.83198016e-16],
       [4.83198016e-16, 9.20164904e-01]])
```

```
     array([[2.93808505e+00, 4.83198016e-16],
            [4.83198016e-16, 9.20164904e-01]])
```

```
pca.explained_variance_
array([2.93808505, 0.9201649 ])
```

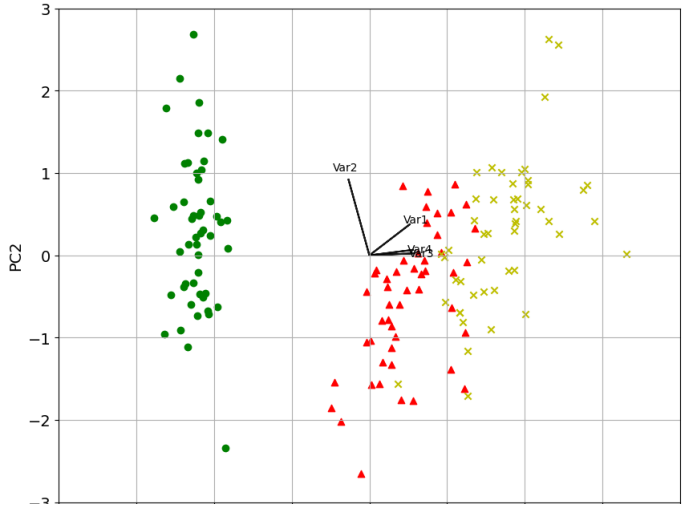
```
     array([2.93808505, 0.9201649 ])
```

```
print(abs( pca.components_ ))
[[0.52106591, 0.26934744, 0.5804131, 0.56485654],
 [0.37741762, 0.92329566, 0.02449161, 0.06694199]]
```

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

```
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] # projection on PC1
    ys = score[:,1] # projection on PC2
    n = coeff.shape[0] # number of 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]) # color based on group
    for i in range(n):
        #plot as arrows the variable scores (each variable has a score for PC1 and one for PC2)
        plt.arrow(0, 0, coeff[i,0], coeff[i,1], color = 'k', alpha = 0.9,linestyle = '-',linewidth = 1.5, overhang=0.2)
        plt.text(coeff[i,0]* 1.15, coeff[i,1] * 1.15, "Var"+str(i+1), color = 'k', ha = 'center', va = 'center',fontsize=10)

    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) # reset ggplot style
# Call the biplot function for only the first 2 PCs
biplot(X_new[:,0:2], np.transpose(pca.components_[0:2, :]), y)
plt.show()
```




```
# Var 3 and Var 4 are extremely positively correlated
np.corrcoef(X[:,2], X[:,3])[1,0]
0.9628654314027957
# Var 2and Var 3 are negatively correlated
np.corrcoef(X[: 1], X[: 2])[1 0]
```

```
np.corrcoef(X[:,1], X[:,2])[1,0]
-0.42844010433054014
```

```
-0.42844010433054014
```

¿Qué es feature importance y para que nos sirve? Calcula la importancia para cada feature de un modelo. Por ejemplo esto nos sirve para darle mayor importancia a datos que el modelo consisere importancias, por ejemplo en la busqueda de un carro, el tipo de carro para ciertas personas puede ser muy importante, por ejemplo si estas buscando Minivans, pues descartas muchos de los vehiculos en le mercado

¿Qué hallazgos fueron los más relevantes durante el análisis del ejercicio? Algo que veo importante al utilizar el PCA es que despues de aplicarlo, las fronteras entre los datos se pueden ver mas definidas, es decir no hay tanta mezcla entre los datos y esto nos hace poder hacer mejor las clasificaciones

¿Dónde lo aplicarías o te sería de utilidad este conocimiento? Pues por ejemplo yo trabajo en seguros, una poliza de seguros en autos tiene un numero de entre 10 y 15 coberturas por lo general, las cueles pueden tener cada una un valor de cobertira, por ejemplo puede tener un deducible de 3,5 o 10% en tus daños, tambien pueden ser valores como monto de gastos medicos que puede ser casi cualqueirmonto pero se estilan brincos de 50 il, como por ejmpo 100, 150 o 200 mil por ocupante... Aplicando un PCa pudieramos ver que coberturas realmente son importantes al precio o prima total de la poliza.