PCA y k-medias para identificar cambios del tipo de suelo en imágenes con base en [1]

Leonardo CA

July 23, 2020

[268]:
```
from numpy import linalg as LA
import rasterio
import numpy as np
import skimage as sk
from skimage.transform import resize
import matplotlib.pyplot as plt
```

El tamaño original de la imagen era de (744, 766) pero la cambie a (720, 720) para hacerla divisible por varios números, en este caso 3, 7 y 9 para probar diferentes tamaños de ventanas. Elegí el INDV porque sé que se usa para clasificación de suelo y trate con 2 imágenes, una de 1996 y otra de 2011 para observar más cambios en la ciudad. Ambas imágenes son del mismo polígono en Hermosillo, Sonora.

[269]:
```
#numero de columnas y filas
nc=720
nr=720
n=nc*nr
#indice normalizado de diferencia de vegetación
indv=[]
#fecha inicial y final
fechai=1994
fechaf=2011
#repeticiones entre fechas
rep=2011-1994
#dif
fevar= (fechaf-fechai+1)
var= int((fechaf-fechai+1)/(rep-1))
for i in range(0,int(fevar),rep):
    fecha=fechai+i
    car=str(fecha)
    print(car)
    ruta="/home/noxd/Home2/hermosillo/data/im" + car +".tif"
    data1=rasterio.open(ruta)
    r = data1.read((3))
```

```
    nir= data1.read((4))
    i_gray = sk.color.rgb2gray((nir-r)/(r+nir))
    i_grayr=resize(i_gray, (nr, nc))
    equalizada = sk.exposure.equalize_hist(i_grayr)
    indv.append( equalizada   )
```

```
1994
2011
```

Aquí se crea la imagen de diferencias.

[271]:
```
dindv=[]
for u in range(0,len(indv)-1):
    dindv.append(abs(indv[u+1]-indv[u]))
```

El algoritmos o función para crear el PCA y la media de los bloques, básicamente usa dos for que recorren la imagen-matriz bloque por bloque, en el caso de la media los promedia y en PCA, calcula la matriz de covarianza, resuelve la ecuación y devuelve los eigenvectores ordenadas y divididos por la raíz de h cuadrado por su respectivo eigenvalor, así como el bloque medio los eigenvalores ordenadas.

[272]:
```
def PCA(dindv,block,indice):
    #contador
    M=0
    bloque_medio= np.zeros(shape=(block*block,1))
    for r in range(0,nr,block):
        for t in range(0,nc,block):
            bloque_medio=bloque_medio+dindv[indice][r:r+block,t:t+block].
 ↪reshape(block*block,1)
            #print(r,t)
            M=M+1
    bloque_medio=bloque_medio/M
    cov= np.zeros(shape=(block*block,block*block))
    for r in range(0,nr,block):
        for t in range(0,nc,block):
            cov=cov+  np.matmul(    dindv[indice][r:r+block,t:t+block].
 ↪reshape(block*block,1)-bloque_medio, dindv[indice][r:r+block,t:t+block].
 ↪reshape(block*block,1).T-bloque_medio.T)
    cov=cov/M
    evalues,evector =LA.eig(cov)
    order = evalues.argsort()[::-1]
    evalues1 = evalues[order]
    evector1 = evector[:,evalues.argsort()[::-1]]
    ev=np.zeros(shape=(block*block,block*block))
    for l in range(0,block*block):
        ev.T[0:block*block,l:l+1]= (1/np.
 ↪sqrt((evalues1[l])*block*block))*evector1.T[0:block*block,l:l+1]
    return ev,evalues1,bloque_medio
```

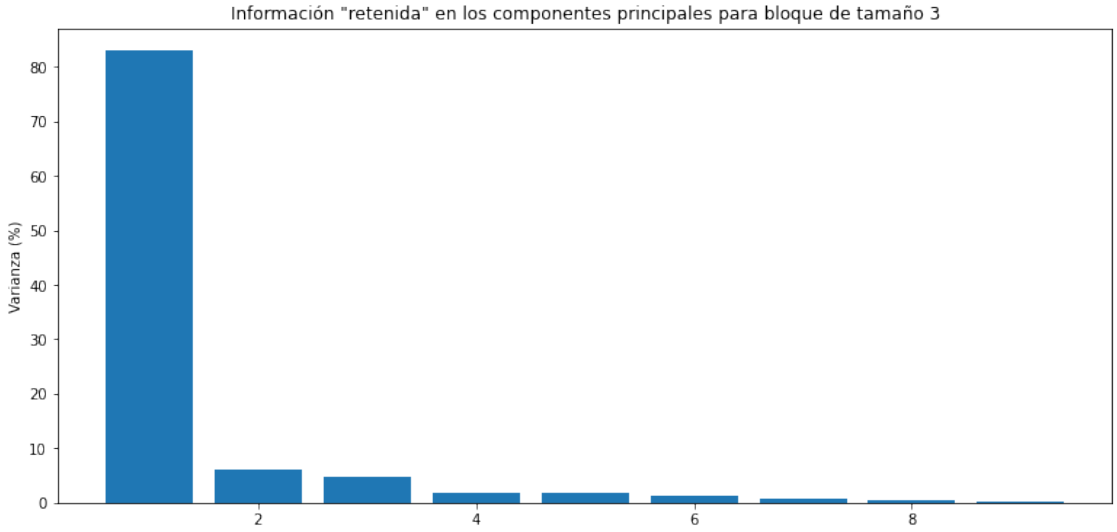
Ya definimos un tamaño de bloque, la variable que dice índice la agrega por si se ocupara hacer analizar más de una imagen de diferencias, que se pueda hacer un for y calcular cambios de varias imágenes de diferencias. Pero aquí solo revise una imagen.

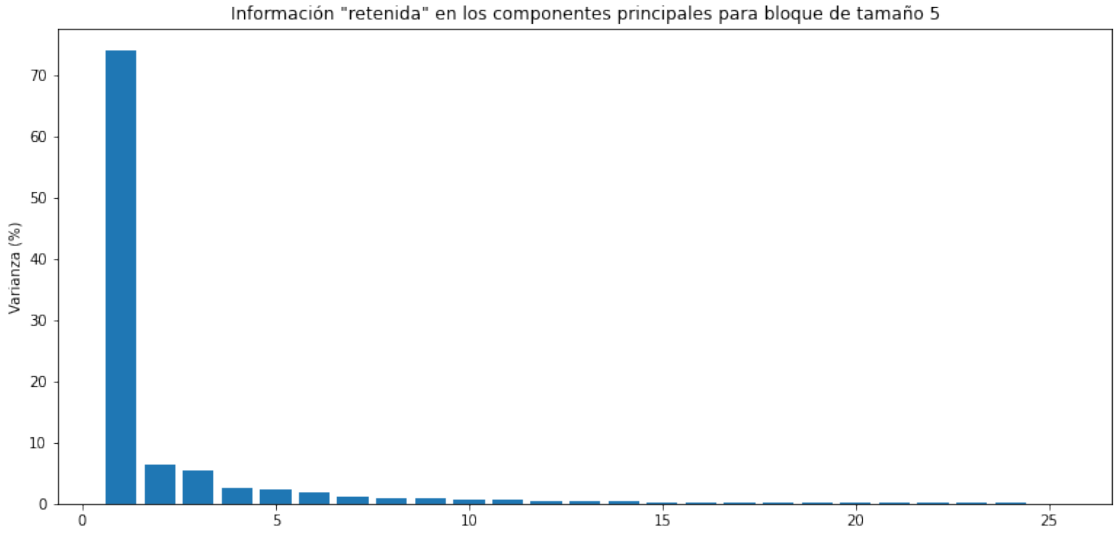
[291]:
```
block=5
indice=0
ev,evalues1,bloque_medio= PCA(dindv,block,0)
```

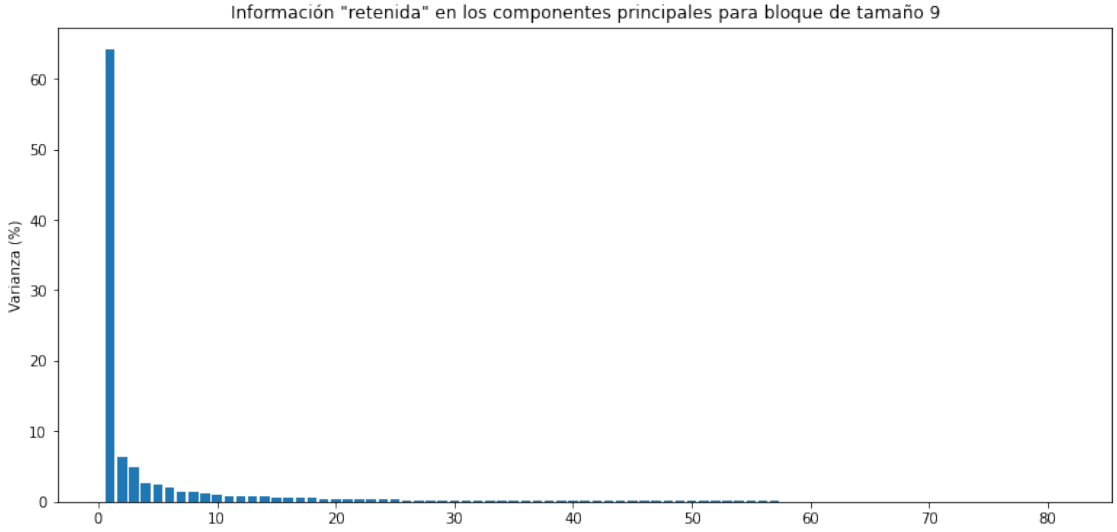
Solo para analizar un poco, grafique el valor de los eigenvalores con respecto a la suma total de los mismos para los diferentes tamaños de bloques (3, 5 y 9), se observa que en general el primer eigenvalor tiene arriba del 60% de la varianza y esta disminuye conforme aumenta el tamaño del bloque, asi como aumenta el número de eigenvalores.

[283]:
```
lista1=[]
lista2=[]
for i in range(0,block*block):
    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 para bloque de␣
 ↪tamaño '+ str(block))
```







El algoritmo o función de k-medias tomo la imagen de diferencias (o más bien la lista que contiene la imagen), el índice de la lista, el número de eigenvalores que se desea usar y el bloque medio. Primero con dos for, se recorre la matriz pixel por pixel creando una ventana centrada en el pixel en el que se está moviéndola masmenos bloque entre dos, a esta ventana se le computal el vector de características que corresponde al pixel central y guardándolo en un diccionario (usando como key el carácter del índice de fila más el carácter del índice de columna) para poder llamarlo durante el proceso de iteración.

Para recorrer el máximo de la matriz posible pero no tener incoherencias en las esquinas, puse un if que solo realice los demás cálculos si el número de filas por el número de columnas de la ventana es igual al cuadrado del número de bloque definido, es decir, si el tamaño de ventana es igual al del bloque.

Después se crean dos vectores de tamaño S con valores aleatorios, se clasifican los vectores con respecto al vector aleatorio con el cual tienen una menor norma, se guardan en dos listas de clases, se calculan nuevos vectores media de cada clase. El algoritmo se detiene cuando la norma de los vectores medios menos los vectores medios anteriores sean menores a 0.0001, deje la norma porque fue la última métrica de error que use pero funcionaba también con la suma del error al cuadrado.

Por último, acomoda los vectores medias con respecto al que tenga la norma menor y regresa estos dos vectores medias.

[277]:
```
def kmeans(dindv,indice,block, S,bloque_medio):
    feature_vector={}
    for r in range(0,nr):
        for t in range(0,nc):
                fv_pixel=dindv[indice][int(r-block/2):int(r+block/2),int(t-block/
 ↪2):int(t+block/2)]
                if fv_pixel.shape[0]*fv_pixel.shape[1] ==block*block:
                    fv=np.matmul(ev.T[0:S,0:block*block]   , fv_pixel.
 ↪reshape(block*block,1)-bloque_medio)
                    feature_vector.update( {str(int(r))+str(int(t)): fv} )
    clase1=[]
    clase2=[]
    media1= np.random.rand(S,1)
    media2= np.random.rand(S,1)
    error1=1
    error2=1
    iterador=0
    while error1>0.0001 and error2>0.0001:
        for r in range(0,nr,block):
            for t in range(0,nc,block):
                fv_pixel=dindv[indice][int(r-block/2):int(r+block/2),int(t-block/
 ↪2):int(t+block/2)]
                if fv_pixel.shape[0]*fv_pixel.shape[1] ==block*block:
                    fv=feature_vector[str(int(r))+str(int(t))]
                    norm1=( np.linalg.norm(fv-media1)    )
                    norm2=( np.linalg.norm(fv-media2)    )
                    if norm1<norm2:
                        clase1.append(fv)
                    elif norm2<norm1:
                        clase2.append(fv)
                    else:
                        flip=np.random.randint(2)
                        if flip==0:
                            clase1.append(fv)
                        else:
                            clase2.append(fv)
        media_anterior1=media1
        media_anterior2=media2
```

```
        media1=0
        media2=0
        for e in range(0,len(clase1)):
            media1=media1+clase1[e]
        media1=media1/len(clase1)
        for z in range(0,len(clase2)):
            media2=media2+clase2[z]
        media2=media2/len(clase2)
        error1=np.linalg.norm(media1-media_anterior1)
        error2=np.linalg.norm(media2-media_anterior2)
        clase1=[]
        clase2=[]
        iterador=iterador+1
    print('Se logro convergencia en', iterador, 'iteraciones')
    prom_media1= np.linalg.norm(media1)
    prom_media2=np.linalg.norm(media2)
    if (prom_media1<prom_media2):
        m1=media1
        m2=media2
        media1=m1
        media2=m2
    elif prom_media2<prom_media1:
        m1=media2
        m2=media1
        media1=m1
        media2=m2
    else:
        print('Las clases no son diferenciables (tienen la misma media)')
    return media1, media2
```

Aquí esta aplicada la función para el último valor de S que use, se ve que puse un print para que nos avisara cuantas iteraciones realizó.

[294]:
```
S=15
media1, media2= kmeans(dindv,indice,block, S,bloque_medio)
```

```
Se logro convergencia en 18 iteraciones
```

La función de mapeo la lista que contiene las imágenes de diferencia, el índice de la imagen, el tamaño de bloque, las dos medias de las categorías del k-medias y el bloque medio. En este caso se recorre la imagen pixel por pixel y la parte del código que dice fv_pixel toma de la matriz una ventana centrada en el pixel en el que se está moviéndola masmenos bloque entre dos.

Para recorrer el máximo de la matriz posible pero no tener incoherencias en las esquinas, puse el mismo if que en k-medias (solo realice el cálculo si el tamaño de ventana es igual al del bloque).

Seguido se multiplica la matriz de eigenvectores (cortándola al número de eigenvectores elegido S) por cada ventana que si cumple con la restricción, se calculan las normas de este vector de características con respecto a cada media que se obtuvo del k-medias y solo si una norma es mayor

a otra se pone 1 a ese pixel, de otra manera se deja como 0. Estos cálculos pude evitarlos si hubiera usado el diccionario de vectores de características del algoritmo de k-medias, pero primero pensé que lo de pixel por pixel se hacía solo en el mapeo, luego pensé que en los dos y ya tenía medio escrito el latex.

[279]:
```
def mapeo(dindv, indice, block,media1, media2,bloque_medio):
    cambio= np.zeros(shape=dindv[indice].shape)
    for r in range(0,nr):
        for t in range(0,nc):
                #print(r,t)
                fv_pixel=dindv[indice][int(r-block/2):int(r+block/2),int(t-block/
↪2):int(t+block/2)]
                if fv_pixel.shape[0]*fv_pixel.shape[1] ==block*block:
                        fv=np.matmul(ev.T[0:S,0:block*block]   , fv_pixel.
↪reshape(block*block,1)-bloque_medio)
                        norm1=( np.linalg.norm(fv-media1)    )
                        norm2=( np.linalg.norm(fv-media2)    )
                        if norm1>norm2:
                            cambio[r:r+block,t:t+block]=1
    return cambio
```

Aquí se llama la función y se obtiene la imagen de cambios del INDV en este periodo.

[295]:
```
cambio= mapeo(dindv, indice, block,media1, media2,bloque_medio)
```

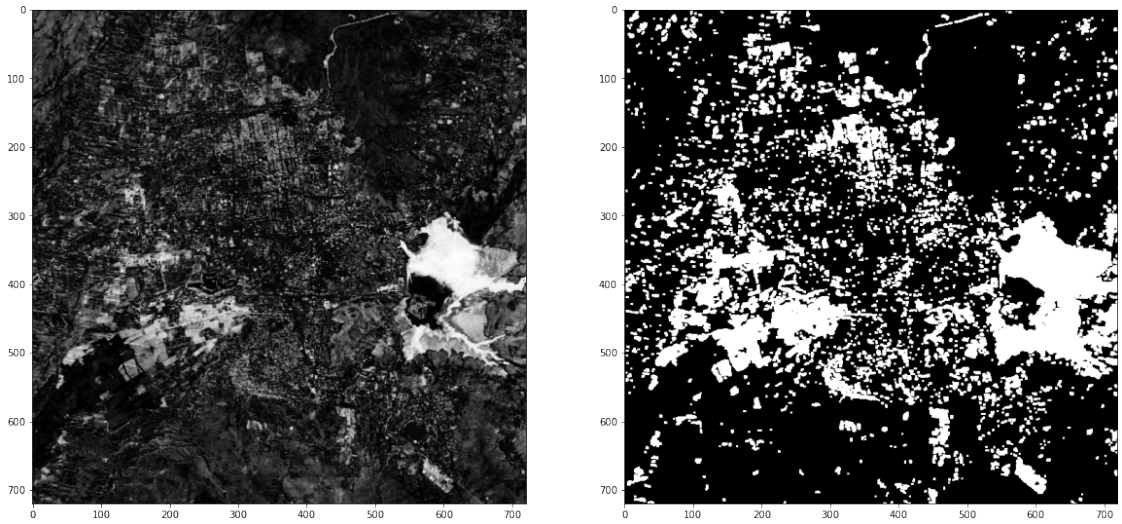
Finalmente grafique la comparación de la imagen de diferencias con los resultados del bloque de tamaño 3, 5 y 8.

0.1 Bloque de tamaño 3

El bloque más pequeño (3) muestra mayor detalle en los cambios puntuales que se dan en estos 15 años, se puede ver que son relativamente bastante los cambios aunque no son muy grandes con la excepción de los que están en el centro a la derecha e izquierda, así como posiblemente al centro un poco para arriba.

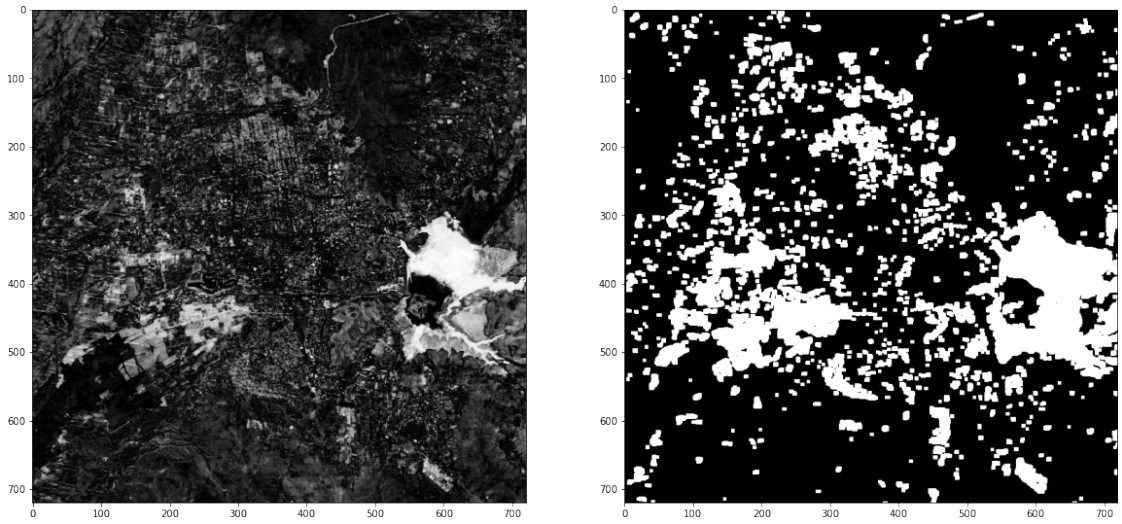
[290]:
```
# Crea figura y ejes (en este caso es un arreglo de 3 x 3)
fig=plt.figure(figsize=(20, 30))
# Hace los ejes accesibles con una sola indexación
columns = 2
rows = 1
fig.add_subplot(rows, columns, 1)
plt.imshow(dindv[0],cmap='gray')
fig.add_subplot(rows, columns, 2)
plt.imshow(cambio,cmap='gray')
```

[290]:
```
<matplotlib.image.AxesImage at 0x7f98fdcb03d0>
```



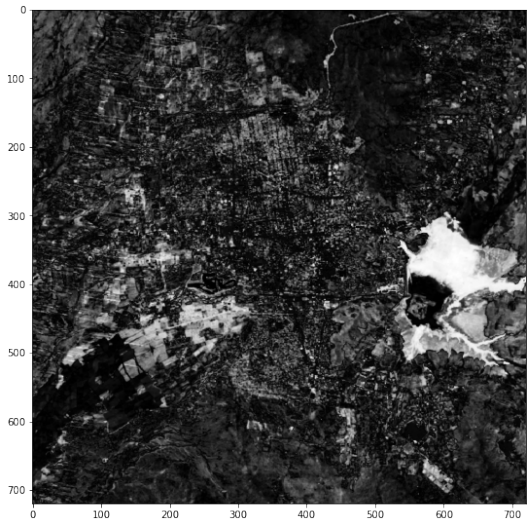
0.2 Bloque de tamaño 5

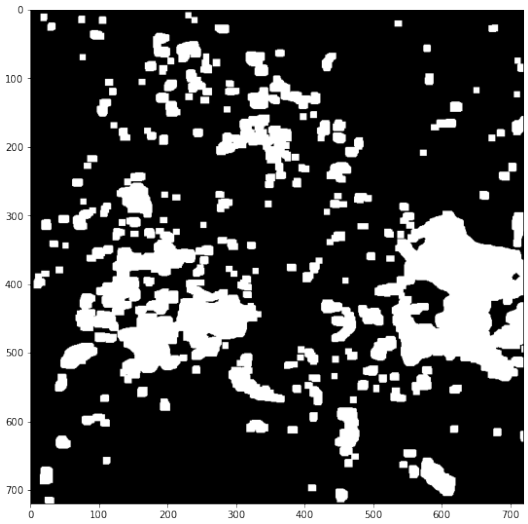
Con un mayor tamaño de bloque (5) se ve que se pierden puntos individuales y algunos de los puntos separados pero cercanos parecen aglomerarse en manchas de cambio mayores, con mayores me refiero a que los cambios se ven más continuos o pegados.



0.3 Bloque de tamaño 9

Por último en el bloque de tamaño 9, se ve que los detalles se pierden más notablemente quedando, me parece, solo aquellas zonas que tienen cambios de magnitud mayor o zonas que parecen cambiar más en conjunto.





References

[1] Turgay Celik (2009) Unsupervised Change Detection in Satellite Images Using Principal Component Analysis and k-Means Clustering. IEEE GEOSCIENCE AND REMOTE SENSING LETTERS. 6(4), 772-776.