Tarea 1 - Parte II

Allan Navarro Brenes
anavarro3106@gmail.com

25/10/2020

1. Eliminación de ruido en imágenes basado en aproximaciones con matrices de bajo rango óptimas

1.1. Resumen del problema

Se tiene una imagen con ruido de la cual se quiere extraer la imagen original sin ruido. Se cuenta con un conjunto de imágenes sin ruido relacionadas con la imagen objetivo y para cada una de estas se tiene una contraparte que presenta un nivel de ruido aditivo de tipo gaussiano. Ambos casos se encuentran en carpetas separadas.

1.2. Formulación matemática

Tomada de [1]
Se tiene un modelo de ruido que puede expresarse de la siguiente manera

$$\mathbf{A}\mathbf{x} + \delta = \mathbf{b} \tag{1}$$

donde $\mathbf{b} \in \mathbb{R}^{\mathrm{n}}$ es la observación, $\mathbf{A} \in \mathbb{R}^{\mathrm{nxn}}$ modela el proceso generador, $\delta \in \mathbb{R}^{\mathrm{n}}$ es el ruido aditivo y $\mathbf{x}$ es la solución deseada.
Se desea obtener una matrix de rango r $\mathbf{Z}$ la cual minimice el error entre la imagen observada y la imagen real del set de entrenamiento, esto puede ser expresado como $\frac{1}{K}\sum_{k=1}^{K}\left\|\mathrm{Zb}^{(k)}-\mathrm{x}^{(k)}\right\|_2^2$. Tomando en cuenta todos los datos de entrenamiento se extrae la matriz que tenga el menor error de la siguiente manera:

$$\hat{\mathrm{Z}} = \arg\min_{\mathrm{rank(Z)}\leq r} \frac{1}{K}\sum_{k=1}^{K}\left\|\mathrm{Zb}^{(k)} - \mathrm{x}^{(k)}\right\|_2^2 \tag{2}$$

Una vez obtenida $\hat{\mathbf{Z}}$, es sencillo obtener una solución a partir de una observación no incluida en el entrenamiento, haciendo una simple multiplicación matriz vector:

$$\mathbf{x} = \hat{\mathbf{Z}}\mathbf{b} \tag{3}$$

Sea K el numero de imágenes de entrenamiento, $\mathbf{B}, \mathbf{C} \in \mathbb{R}^{\mathrm{nxK}}$ matrices que contienen las imágenes con ruido y sin ruido respectivamente. Sea $\widetilde{\mathbf{V}}$ la matriz de vectores singulares por la derecha de B. Se define $\mathbf{P} = \mathbf{C}\widetilde{\mathbf{V_s}}\widetilde{\mathbf{V_s}}^T$ con $s = rank(\mathbf{B})$ con esto se obtiene $\hat{\mathbf{Z}}$

$$\hat{\mathbf{Z}} = \mathbf{P}_r\mathbf{B}^{\dagger} \tag{4}$$

donde $\mathbf{P_r}$ es la matriz de rango r calculada a partir de la SVD de $\mathbf{P}$, el rango $r \geq rank(P)$ ó $1 \leq r \leq rank(P)$, $\mathbf{B}^{\dagger}$ es la pseudoinversa de $\mathbf{B}$. Para esta tarea se utilizan valores de r=[40,120,220,300,380,416]

1.3. Pasos a realizar para la implementación

Se utiliza el lenguje python.

1. Importar paquetes a utilizar

```
import cv2
from os import listdir
from os.path import isfile, join
import numpy as np
import matplotlib.pyplot as plt
```

2. Leer la imagen a limpiar y obtener sus dimensiones

```
imagen_a_limpiar=cv2.imread("limpiar.jpg",cv2.IMREAD_GRAYSCALE)
shape = imagen_a_limpiar.shape
rows=shape[0] #columnas de la imagen
cols=shape[1] #filas de la imagen
```

3. Se crea una lista con los archivos a cargar en las carpetas 'ruido' y 'original' y se crea una matriz que almacena estas imagenes (B, C) lista con el nombre de las imagenes originales

```
imgs_ruido= [join('.','ruido',f) for f in listdir('ruido') if isfile(join('ruido',f))]

#lista con el nombre de las imagenes con ruido
imgs_orig= [join('.','original',f) for f in listdir('original') if isfile(join('original'
    ,f))]

#verifica que el numero de imagenes sean iguales en ambos directorios
if len(imgs_ruido) != len(imgs_orig):
    print('no coincide la cantidad de imagenes original con las imagenes con ruido')
    exit(-1)

#C: imagenes originales
C_orig=np.zeros((rows*cols,len(imgs_orig)))
#B: imagenes con ruido
B_ruido=np.zeros((rows*cols,len(imgs_ruido)))
```

4. Se iteran las listas con los nombres de los archivos y se almacenan en B y C, cada imágen es vectorizada

```
#carga las imagenes en la matriz correspondiente
for index in range(len(imgs_orig)):
    #trae los nombres de los archivos de la lista
    img_ruido_path=imgs_ruido[index]
    img_orig_path=imgs_orig[index]

    #carga las imagenes y las vectoriza
    B_ruido[:,index]=cv2.imread(img_ruido_path,cv2.IMREAD_GRAYSCALE).reshape(rows*cols)
    C_orig[:,index]=cv2.imread(img_orig_path,cv2.IMREAD_GRAYSCALE).reshape(rows*cols)
```

5. Se calcula la SVD de B y su rango, luego se calcula $\mathbf{P}$, su SVD y $\mathbf{B}^{\dagger}$

```
# SVD de matriz B
u_b,s_b,vh_b=np.linalg.svd(B_ruido,False)

eps=np.finfo(s_b.dtype).eps #lo que es considerado 0 por python

#rango de B, valores singulares mayores a eps
rango_b=len(s_b[s_b>eps])

#calcula Vs
Vs=vh_b.transpose()[:,0:rango_b]

#calcula P
P= C_orig@Vs@Vs.transpose()

#Pseudo inversa  B
B_pinv= vh_b.transpose() @ np.linalg.inv(np.diag(s_b)) @ u_b.transpose()

#SVD P
u_p,s_p,vh_p=np.linalg.svd(P,False)
```

6. Se iteran los rangos deseados y se calcula el resultado de la imagen según 3

```
#itera los rangos deseados
rangos=[40,120, 220, 300, 380, 416] #rangos especificados en el enunciado
pos=0
for r in rangos:
```

```
    pos +=1
    #calcula P reducida con el rango r
    Pr= u_p[:,0:r] @np.diag(s_p[0:r]) @ vh_p.transpose()[:,0:r].transpose()

    #Calcula la apriximacion de Z
    Zt=Pr@B_pinv
    # reconstruccion de la imagen
    reconst=np.dot(Zt,imagen_a_limpiar.reshape(rows*cols))

    reconst=reconst.reshape(rows,cols)
    #si hay numeros negativos, se ponen en 0, los mayores a 255 se ponen en 255
    reconst[reconst<0]=0
    reconst[reconst>255]=255
    #cambio de formato a uint8
    reconst=reconst.astype(np.uint8)

    #se guarda la imagen
    #cv2.imwrite('limpia_r'+str(r)+'.jpg',reconst)
    plt.subplot(2,3,pos)
    plt.imshow(reconst,cmap='gray')
    plt.title('limpia r: '+str(r))

plt.show()
```

1.4. Resultados

Se obtienen las siguientes imágenes con distintos rangos

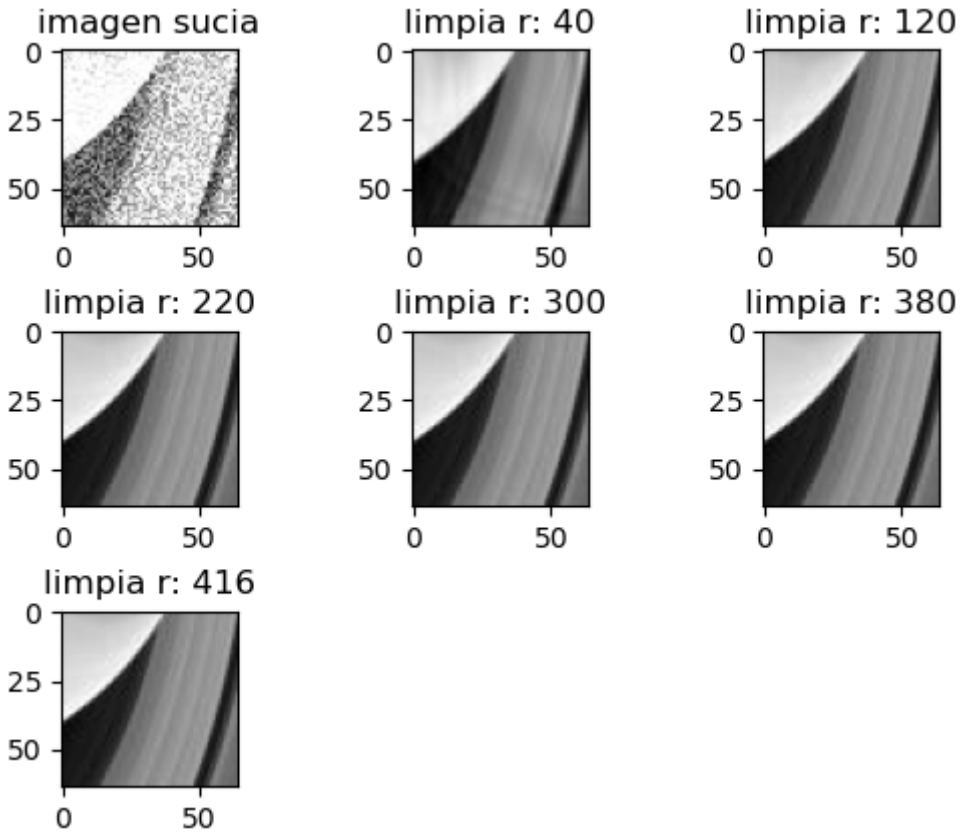

Figura 1: Imágenes reconstruidas con distintos valores de r

Referencias

[1] J. Chung y M. Chung, "Computing optimal low-rank matrix approximations for image processing," en *2013 Asilomar Conference on Signals, Systems and Computers*, 2013, págs. 670-674. DOI: 10.1109/ACSSC.2013.6810366.