Elementos de Cálculo Numérico / Cálculo Numérico

Primer Cuatrimestre 2020

Séptimo ejercicio computacional
Lunes 22/06/20 al Lunes 29/06/20
Recuerde subir el archivo en formato ejercicioX_NOMBREAPELLIDO.py
Recuerde enviar su código al hacer consultas

En este ejercicio exploraremos una de las aplicaciones del método de descomposición en valores singulares (SVD). Este consiste en resumir información, reduciendo la cantidad de espacio en la memoria para guardar un objeto a cambio de pérdida de información. En la teórica vimos que podíamos descomponer $A \in \mathbb{R}^{n\times m}$ en tres matrices $U \in \mathbb{R}^{n\times m}$, $S \in R^{m\times m}$ y $V \in \mathbb{R}^{m\times m}$, de forma tal de poder rearmar $A = USV^T$. Desde el punto de vista de la compresión de imagenes, es más conveniente en cambio cargar el cambio de dimensión en S, de forma de tener $U \in \mathbb{R}^{n\times n}$, $S \in R^{n\times m}$ y $V \in \mathbb{R}^{m\times m}$, donde lo que hacemos es agregar filas con ceros a S, de forma tal que ya no es una matriz cuadrada. Como ya sabemos que S es diagonal, en realidad podemos almacenar sólo su diagonal, y las dimensiones pertinentes, ahorrando espacio. En esta lógica, la compresión se realiza reduciendo la cantidad de elementos de la diagonal que preservamos de S. La idea es que a medida que incorporamos más elementos de la diagonal, más información mantendremos de la imagen. Reducir el tamaño de S ahorra memoria porque nos permite reducir la cantidad de elementos de U y V que necesitamos guardar (al haber tantos ceros en S, muchos valores no serán relevantes).

Mucho más modestamente, el objetivo de este ejercicio es explorar como cambia la calidad de la compresión a medida que incorporamos más autovalores a la imagen.

Para esto:

A Emplearemos la función `svd` de modulo de álgebra lineal de numpy:

```
import numpy.linalg as npl
npl.svd()
```

Esta función retorna tres arrays, `U,s,V`, de forma tal que podemos recuperar la matriz `A` de la siguiente forma:

```
A = np.array([[1,2,3],[4,5,6]])
U,s,V = npl.svd(A)
S = np.zeros((A.shape[0], A.shape[1]))
S[:len(s), :len(s)] = np.diag(s)
A_ = np.dot(U,np.dot(S,V))
print(np.mean(A_-A))
```

No esperen que de cero (error de punto flotante), pero el error es muy chiquito. Exploren los tamaños de cada una de estas tres matrices, usando el comando `X.shape`, donde `X` es la matriz.

Nota: En general, para una dada dimension de un array `x[a:b:c]` tomará los elementos de `x` entre `a` y `b-1`, espaciando cada `c` pasos. Si queremos ir hasta el final del array, podemos omitir b, y si queremos mantener todos los elementos, sencillamente ponemos un `:`.

B Cargue y grafique la imagen `arbol.jpg` provista:

```
import imageio as img
image = img.imread('arbol.jpg',format='jpg')
```

Esto importará la imagen como una matriz de $n \times m \times 3$, donde esa tercer dimensión corresponde con los colores de la matriz. Construya una función que convierta la imagen a escala de grises, como un promedio pesado entre las tres matrices de cada color, siendo r el peso del rojo, g el del verde (y el peso del azul queda definido $b = 1-r-g$). Aproveche el siguiente modelo:

```
def a_grises(image,r,g):
    R = image[:,:,0] # Matriz de rojos
    G = image[:,:,1] # Matriz de grises
    B = image[:,:,2] # Matriz de azules
    gris = ## COMPLETAR ##
    return(gris)
```

Compare el resultado de su graficando la imagen resultante. Pruebe los coeficientes `r=g=1/3`, y `r=0.3,g=0.59`. Para graficar, si `A` es un array:

```
plt.imshow(imagen_gris,cmap='gray',vmin=0,vmax=255)
plt.savefig('ejemplo_en_escala_de_grises.jpg')
```

El comando `cmap` sirve para indicar el mapa de colores, y `vmax,vmin` indican el rango de valores para el máximo y mínimo de intensidad

C Pruebe sobre esta matriz la descomposición en valores singulares y compare ambas imagenes. ¿Es satisfactorio el resultado?

D Construya una función que tome la matriz `A`, aplique SVD, convierta en cero el último `p` porcentage de elementos de `s`, y devuelva una matriz `A_`, resultante de recomponer la matriz con sólo `1-p` porcentaje de los autovalores:

```
def reduce_svd(A,p):
    U,s,V = ## COMPLETAR ##
    n_elementos = int(p*len(##COMPLETAR))
    s[#COMPLETAR#] = 0
    S = np.zeros((A.shape[0], A.shape[1]))
    S[:len(s), :len(s)] = np.diag(s)
    A_ = ## COMPLETAR ##
    return(A_)
```

Pruebe el resultado de aplicar esta función a la imagen `arbol.jpg` manteniendo un 10% de los autovalores de la matriz (es decir, convirtiendo en cero un 90% de los elementos de `s`).

E Implemente un programa que calcule el error promedio en la imagen en función del porcentaje de autovalores que elimina:

```
error = []
for p in ## COMPLETAR ##:
print('Calculando con p='+str(p))
    image_ = reduce_svd(image,p)
    error.append(np.mean(np.abs(image_-image))/np.mean(image))
```

¿Cuantos autovalores considera necesarios para aproximar la imagen? Compare su criterio visualmente (graficando algunas imagenes) con lo que observa del gráfico.

F Repita empleando las otras imagenes provistas. ¿Cambia el resultado con la complejidad de la imagen? Grafique todas las curvas de error en una misma imagen y compare.

Nota: Este paso puede demorar. Una buena práctica es incorporar contadores que muestren el progreso, para saber que el código va a avanzando y a qué velocidad.