Clusterizador de imágenes

Luca Manzi
Ingeniería Informática
ETSII
Sevilla

10-02-2021

1 Objetivo

Crea un clusterizador de imágenes (agrupa las imágenes de un dataset según su parecido). Nota: busca un dataset adecuado de imágenes, ya clasificadas y de pequeño formato.

1.1 Introducción

Para crear un clusterizador, primero debemos entender qué características de las imágenes tenemos que analizar, y esto depende principalmente del tipo de imágenes que queremos clusterizar, por ejemplo, se puede utilizar una comparación de forma o colores, patrónes u otras características. A raiz de lo anteriormente comentado necesitamos comparar un conjunto de características y medir la diferencia entre las imágenes, y cada imagen con n características; esto será como hacer una resta de vectores n-dimensionales. Luego comenzamos a estandarizar las imágenes, es decir, guardándolas con la misma forma, para que podamos dividirlas en sectores y poder comparar los distintos sectores correspondientes.

1.2 Dataset

Empezamos con el conjunto de datos, para este proyecto decidí usar un subconjunto de un dataset de flores, del cual podemos encontrar muchos proyectos de inteligencia artificial, con fotos de muchos tipos de flores diferentes. La imagen 2 lo resume.

Otro conjunto de datos que utilicé es una colección de ladrillos Lego de colores neutros, tomados desde diferentes ángulos

1.3 Preprocesamiento

Las imágenes no tienen la misma forma ni la misma definición y por lo tanto sería apropiado estandarizarlas. Necesitamos una lista con los nombres de las mismas, para poder referirnos a ellas y su contenido y así reconocer las Clúster al que pertenecen por dicho nombre y facilitar el proceso. Comenzamos a extraer todos los nombres de las imágenes que queremos agrupar, podemos hacerlo automáticamente con un script como el siguiente:

```
import os

os.path.abspath('')
file_list = os.listdir(os.path.abspath(''))
print (file_list)
file_list.sort()
file_object = open(r"names.txt", "w+" )
for elem in file_list:
    if elem.endswith('jpg')or elem.endswith('png'):
        print (elem)
        file_object.write(elem + '\n')
file_object.close()
```

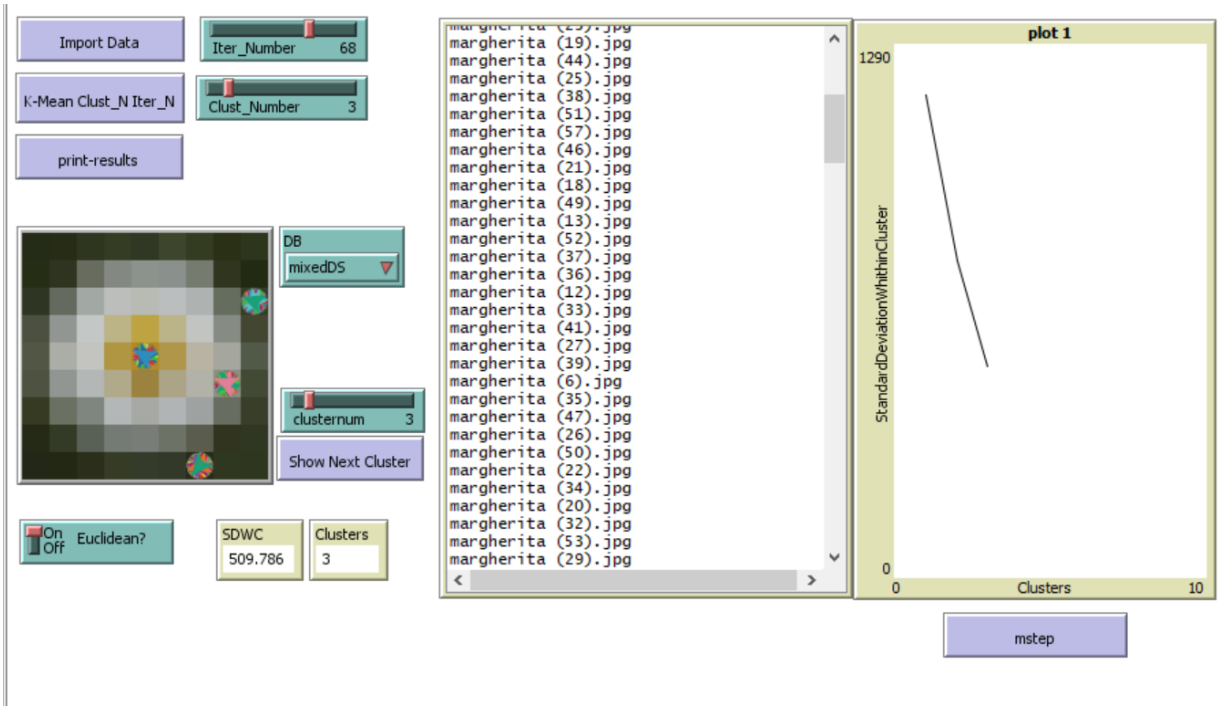

Figure 1: Interfáz
La interfaz en este caso necesita alguna explicación:

- Import Data - importar datos de la base de datos en DB.
- K-Mean Clust_N Iter_N - hace un solo intento de k-mean con los valores de los controles deslizantes a su derecha.
- print-results - imprimir en el box los clústeres con los elementos contenidos.
- Show Next Cluster - muestra la "posición" de cada cluster en forma de colores.
- mstep - realiza la operación k-means a partir de 1 grupo hasta el número máximo indicado en el control deslizante, y dibuja el gráfico de su desviación estandár.

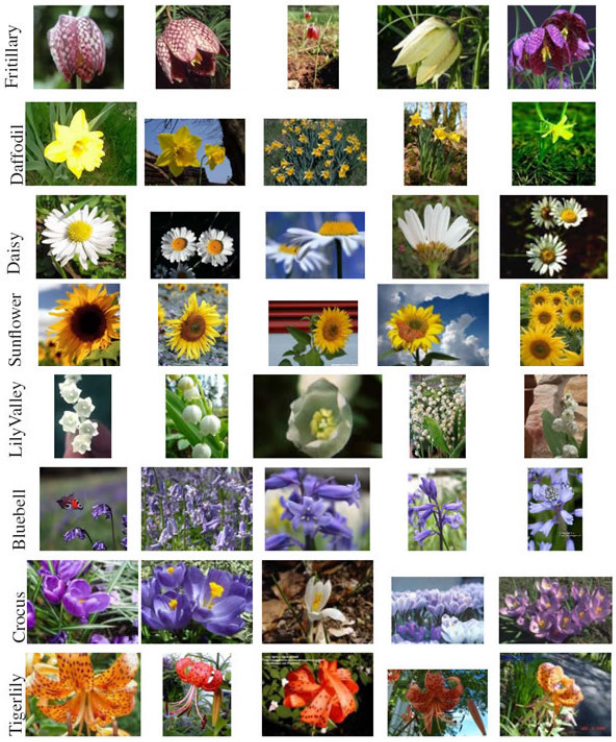

Figure 2: Conjunto de datos descargado de https://www.robots.ox.ac.uk/~vgg/data/flowers/17/index.html

Desde aquí podemos usar otro script para recortar cada imagen en un cuadrado de tamaño máximo, comenzando desde el centro, donde es más probable que este el contenido principal de la foto. Hemos elegido un tamaño único para evitar deformaciones. Acontinuación establecemos una resolución, manteniendo los colores sin cambios, en este script 300×300.

```
import cv2
import os

def center_square_crop (img):
        height, width = img.shape[:2]
        crp = int(min(height, width)/2)
        xmedio, ymedio = int(width/2), int(height/2)
        cropped = img[ymedio-crp:ymedio+crp, xmedio-crp:xmedio+crp]
        return cropped

def center_square_scale(img, pixel=300):
    if img.shape[0]!=img.shape[1]:
        return False
    factor = pixel/img.shape[0]
    return cv2.resize(img,(int(img.shape[1]*factor), int(img.shape[0]*factor)))

dir = os.path.dirname(os.path.abspath(__file__))
imgout = os.path.join(dir, 'dataset')
if not os.path.exists(imgout):
    os.mkdir (imgout)
print (dir)
filein = os.path.join(dir, 'names.txt')
file1 = open(filein, 'r')
Lines = file1.readlines()
for line in Lines:
    if (line.endswith('.jpg\n') or line.endswith('.png\n')):
        imgin = os.path.join(dir, line.strip())
```

```
        img = cv2.imread(imgin)
        new_img = center_square_scale(center_square_crop (img))
        imgout = os.path.join(dir, 'dataset', line.strip())
        success = cv2.imwrite(imgout, new_img)
        if success == True:
            print ('\b'*200 + line.strip() + " saved successfully", end='')

print('\b'*200 + "Everything went smooth, \nConversion complete!")
```

2 Desarrollo

2.1 Representar una imagen

Para la representación de cada imagen individual elegimos agrupar los píxeles en diferentes grupos, por ejemplo, dividiendo recursivamente la imagen en sectores y guardando el color promedio de ese sector en una variable. De esta forma podemos realizar la comparación y, por tanto, la resta de los sectores correspondientes.

El color se representará como un vector tridimensional: R(rojo), G (verde) y B(azul) y la representación de una imagen será un vector n dimensional de vectores tridimensionales, donde n es el número de sectores que queremos usar como parámetro de la división:

$$\mathbf{Image_{vector}} = \left\{ \begin{bmatrix} R_1 \\ G_1 \\ B_1 \end{bmatrix} \begin{bmatrix} R_2 \\ G_2 \\ B_2 \end{bmatrix} \begin{bmatrix} R_3 \\ G_3 \\ B_3 \end{bmatrix} \quad \cdots \quad \begin{bmatrix} R_n \\ G_n \\ B_n \end{bmatrix} \right\}$$

Por lo tanto, la diferencia de color entre dos áreas a b vendrá dada por la longitud $|\overrightarrow{D}|_{a,b}$ de la diferencia entre el vector $\begin{bmatrix} R_a & G_a & B_a \end{bmatrix}$ e $\begin{bmatrix} R_3 & G_3 & B_3 \end{bmatrix}$

Visualizaremos cada píxel como un triángulo, partiendo del centro llegamos a un máximo de 255 para cada color. Utilizar otros métodos es muy poco práctico. Por ejemplo, una medida de color unidimensional, como un promedio, nunca nos dará un resultado sobre el color sino sobre el complejo de intensidad de un determinado color, como puede ser un rojo fuerte o un verde fuerte el algoritmo los indentificará como idénticos.

Ni siquiera podemos usar otra escala, como la unidimensional de netlogo (en la 3) porque en este caso dos colores muy parecidos como el 9,9 y 139,9 resultan muy lejanos cuando vamos a medir la distancia entre ambos.

2.2 Importación

Como ya tenemos la función para importar imágenes en netlogo manteniendo el color en formato rgb, podemos configurar el número de píxeles de la interfaz principal que tomará el valor **n**. Entoces leemos el archivo de texto creado anteriormente, y recorremos cada nombre de cada imagen para cargarla en la interfaz y luego almacenarla en una tortuga *punto*, asignando también a su propriedad *name* la línea que acabamos de leer, esto será útil para ver rápidamente el contenido de cada clúster.

Al final la estructura de *punto* va a ser de la forma:

[position cluster name]

donde position representa su "posición" vectorial, cluster es el conjunto al que pertenece (vacío por ahora) y name es el nombre del fichero de origen.

```
to import_data
  clear-all
  let data []
  let currentd  "C:\\Users\\ronin\\Documents\\UNI\\Programming\\AI\\ClusteringProject\\
    ImageClusterization\\"
  ;;DB is the database in a folder that i can choose from Interface
  set-current-directory word currentd DB
  file-open "names.txt"
```

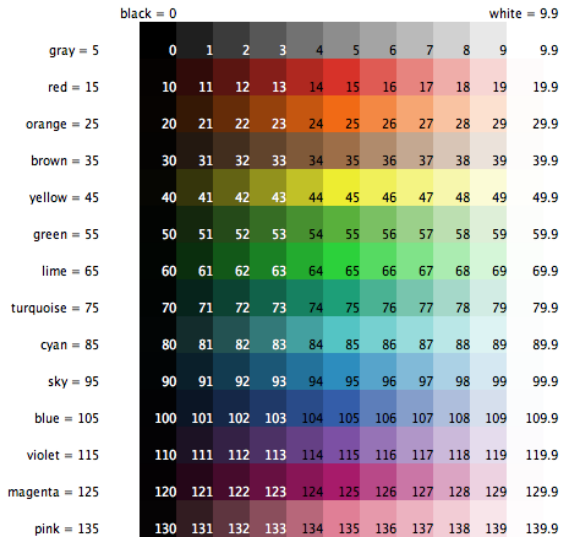

Figure 3: Fuente http://ccl.northwestern.edu/netlogo/docs/programming.html#colors

```
  while [not file-at-end?][
    set data file-read-line
    import-pcolors-rgb data
    create-points 1 [
      set pos gen
      set name data
      ]
  ]
  file-close
end

to-report gen ;; generates and reports the color vectorization
  let b map [x ->
    ([pcolor] of x ;;mean
  )] sort patches
  report (b)
end
```

2.3 Distancia

A continuación escribiremos la función de distancia, que no necesariamente tiene que ser única. Por ejemplo, la distancia estándar euclidiana y la distancia redmean, que es la más cercana a la distancia de la percepción humana.

```
to-report distancia [p1 p2]
  ifelse euclidean? = true[
    report sqrt sum (map [ [x y] -> (eucl x y) ^ 2 ] p1 p2)
  ][
    report sqrt sum (map [ [x y] -> (redmean x y) ^ 2 ] p1 p2)
  ]
end

to-report eucl [p1 p2]
  report sqrt sum (map [ [x y] -> (x - y) ^ 2 ] p1 p2)
end

to-report redmean [p1 p2]
  let r (first p1 + first p2) / 2
  report sqrt (
    (2 + (r / 256)) * ((first p1 - first p2)^ 2)
```

```
     + 4 * ((first bf p1 - first bf p2)^ 2)
     + (2 + (255 - r) / 256) *  ((last p1 - last p2)^ 2)
    )
end
```

3 K-mean

3.1 Algoritmo

El algoritmo k medias permanece muy cercano al original n-dimensional, estableciendo aleatoriamente un número **k** de puntos, en cuya posición se crea un cluster; a cada punto se le asignará el cluster más cercano y se calculará el promedio, es decir, el centro del grupo es temporal así formado. Todo esto se repitirá un número *Iter* de iteraciones, ajustando cada vez más los clusters.

```
to crea-clusters [K] ;; create K random clusters (from points)
  ask clusters [die]
  ask n-of K points [
    hatch 1 [
      set breed clusters
    ]
  ]
end

to k-medias [K Iter] ;; main function
  crea-clusters K
  repeat Iter [
    K-medias-step
  ]
end

to K-medias-step
  ask points [  ;; Seleccionamos para cada Point el cluster mas cercano
    let p-point pos
    set cl min-one-of clusters [distancia pos p-point]
  ]
  ask clusters [   ;; Actualizamos la posicion del cluster al punto medio de sus puntos
    set pos centro ([pos] of points with [cl = myself])
  ]
end
```

La única diferencia está en tener una distancia de un vector de vectores, por lo que el centro se obtiene calculando el centro de cada coordenada tridimensional, al final será como tener una imagen con los colores medios de todas las imágenes del grupo.

```
to-report centro [lista-pos] ;; return center of a 3 x n matrix
  let d length first lista-pos
  let indices (range d)
  let ret map [ x ->
    map [y ->
      mean coords y coords x lista-pos
    ] (range 3)
  ] indices
  report ret
end
```

También se puede imprimir esta "imagen promedio" de cada grupo en la pantalla con la función de *stampa* o con el botón apropiado en la interfaz principal, así para tener una idea de cómo se comporta cada grupo y dónde están las diferencias a simple vista.

```

to stampa [k] ;; print cluster color characteristics
  ifelse (k < length sort clusters)[
    set k item k sort clusters
    (foreach ([pos] of k) (range length sort patches)[ [c p] ->
      ask item p sort patches [set pcolor c]
```

```
      ]
    )
  ][
    ask patches [set pcolor black]
  ]
end
```

3.2 Medidas de dispersión

Para evaluar el comportamiento del algoritmo tenemos que encontrar medidas adecuadas, por ejemplo puedo usar la varianza o la desviación estándar que se define como el cuadrado de la suma total de todas las diferencias de los puntos del centro, ambas serían efectivas ya que trabajamos con una sola escala de medida. Entonces calculamos la desviación estándar, que es la raíz de lo anterior. Matemáticamente, si pongo n = número de imágenes en el grupo, x_i = posición de la i-ésima imagen, μ = posición del cluster, se puede describir

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n}}$$

```
to-report clvariance [ x ]  ;; calculate standard deviation in cluster x
  let cld sum map [ p ->
    (distancia p ([pos] of x)) ^ 2
    ][pos] of points with [cl = x]
  report precision (sqrt (cld /(length sort points with [cl = x]))) 3
end
```

Y sumo todas las desviaciones dividiéndolas por el número de clusters para tener un índice general de comportamiento, una desviación media total que llamo SDWC, Standard Deviation Whithin Clusters:

```
to-report calc-SDWC ;; calculate standard deviation in total
  let temp 0
  foreach sort clusters [ cc ->
      set temp temp + clvariance cc
    ]
  set KK length sort clusters
  let t2 (temp / length sort clusters)
  print (list "clusters = " length sort clusters "SDWC" (precision t2 3)   "Standard
    deviation total" (precision temp 3) )
  report t2
end
```

4 Buscar el mejor K

Finalmente, tenemos todas las herramientas para encontrar el mejor k, es decir, ver cuál es el número óptimo de clústeres en los que dividir el conjunto de datos.

Una solución podría ser precisamente la de minimizar la desviación media (SDWC), pero habría una contradicción ya que la desviación media tiende a cero.

Siendo así, cuando k = número de puntos $\rightarrow SDWC = 0$ siendo la distancia de cada punto al clúster generado en sí mismo = 0.

4.1 El método del codo

Sin embargo, se puede analizar el gráfico para ver dónde se aplana más la función $SDWC(data, k)$, es decir, comienzamos a observar muy pocas mejoras al aumentar k y se aprecia una forma con similitud a la de un codo.

Escribimos la función mstep que intenta agrupar varias veces con k que varía de uno a n, e imprime cada vez el SDWC mínimo que ha encontrado, para evitar cualquier mínimo local y tener el gráfico lo más real

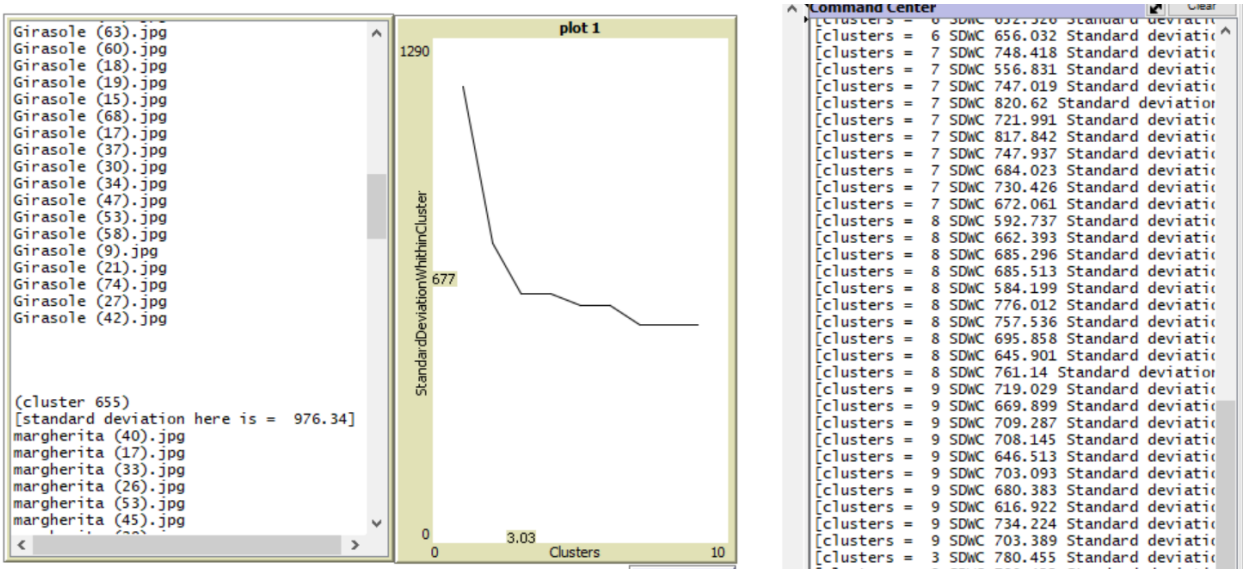

Figure 4: Gráfico obtenido con el conjunto de datos *mixed*

posible. En la figura 4 vemos claramente que el conjunto de datos se puede dividir en 3 clústeres y que incluso se añadan muchos grupos adicionales, por ejemplo 7 8 y 9, el mejor valor de SDWC se mejoraría minimamente.

En el otro caso de 5 usamos la base de datos de flores, y vemos que el valor recomendado de K es el valor 7 incluso si el número de especies es 9. Esto se debe a que a pesar de que hay 3 clusters agrupados en uno, los tres tienen en común un predominio del color verde en las imágenes, aumentando el número de clusters y luego disminuyendo cada vez menos la SDWC.

4.2 Método genético

También se podría usar una solución optimizada, por ejemplo, usando un algoritmo genético. El cual vamos a crear un número arbitrario de posibles clusters, y vamos a generar la población de las soluciones con los clusters usados, estableciendo una cadena de 0 o 1 dependiendo de si usamos o no ese clúster en particular en la configuración.

Un ejemplo de posibles configuraciones iniciales puede ser como en la figura 6

Y la función de inicialización tiene la forma:

```
to genetically-best-k
  foreach  bf(range (Clust_Number + 1)) [ c -> ;; create possible clusters
    foreach (range (5))[
      k-medias c Iter_Number
      let startingzeroes length sort possibleclusters
      foreach sort clusters [ x ->
        create-possibleclusters 1 [ set pos [pos] of x ]
      ]
      create-sols 1 [ ;; create population
        set bits n-values startingzeroes [0]
        set bits (sentence bits (n-values (length sort possibleclusters - startingzeroes)
    [1]))
        ]
    ]
  ]
  ask sols [ ;;fill with zeroes to have a bit for every cluster, also the ones created later
    let fillingzeroes (length sort possibleclusters - length bits)
    set bits sentence bits (n-values fillingzeroes [0])
  ]
  print "done"
end
```

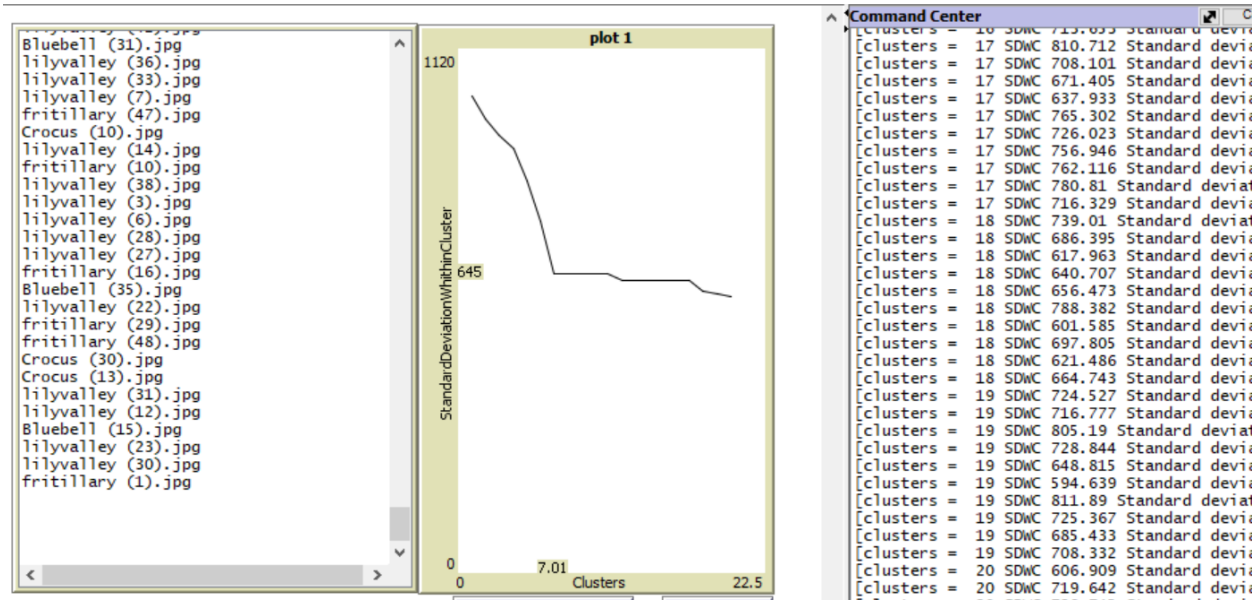

Figure 5: Gráfico obtenido con el conjunto de datos *flowers*

Lo que haremos es generar diferentes soluciones posibles a partir de los estados iniciales calculados, combinando los diferentes clusters según su *fitness*.

Para optimizar la función *fitness*, podríamos utilizar fácilmente la minimización de SDWC, obteniendo los mismos resultados, es decir, al aumentar los clústeres mejoraría.

Por tanto para no encontrarnos con el mismo problema podemos buscar otra definición, la de *silhoutte* que se define como la distancia mínima de este punto desde el punto más cercano de otro grupo menos la distancia promedio de un punto a todos los demás en el su mismo clúster; todo dividido por el máximo entre los dos valores. De esta forma el valor de la silhouette permanece bloqueado entre -1 y 1; y la distancia al punto más cercano de otro conjunto juega un papel importante para mantener el número de clústeres lo más bajo posible. Sin embargo, esta operación es muy compleja a nivel computacional.

```
observer> foreach (range 15) [i -> show [bits] of item i sort sols]
observer: [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
observer: [0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
observer: [0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
observer: [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
observer: [0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
observer: [0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
observer: [0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
observer: [0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
observer: [0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
observer: [0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
observer: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0]
observer: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0]
observer: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0]
observer: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0]
observer: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1]
```

Figure 6: Configuración genética de las soluciones: representación de los clústeres posibles utilizados

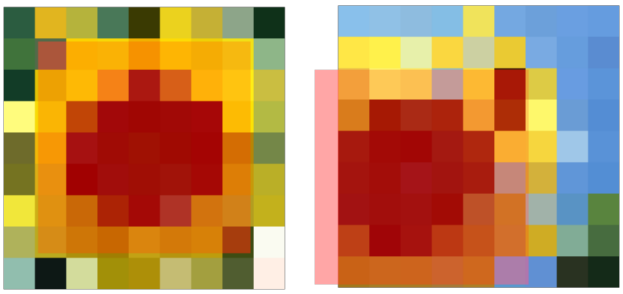

Figure 7: En rojo los pixeles que vamos a usar para obtener la distancia menor

5 Ideas

5.1 Discartar variables

Podriamos intentar usar una solución más efectiva agregando algunas operaciones. Por ejemplo, podría analizar no la imagen completa en sí sino un grupo de píxeles centrales de una imagen y realizar la operación de diferencia solo con ese grupo, tratando de colocar el centro de la diferencia en diferentes posiciones, de tal modo, tomamos como diferencia final la mínima entre las diferencias probadas. De esta forma, eliminamos el problema de tener dos imágenes iguales pero descentralizadas (figura 7), que para nuestro propósito deben tener diferencia cero, pero sin esta operación dan una diferencia > 0. Sin embargo, esta solución sigue siendo de tipo fuerza-bruta, porque se pueden encontrar soluciones más eficientes utilizando patrones u otros.

Podrías pensar en otros métodos para clasificar cada foto, el color dominante después de alguna operación. Por ejemplo si tuvieras un conjunto de fotos de objetos en el mar, ignorar el azul o, en cualquier caso, realizar una operación en particular.

Todo esto es completamente dependiente del conjunto de datos. Por consiguiente, la solución más avanzada sigue siendo utilizar un reconocimiento de patrones y luego profundizar aún más con la inteligencia artificial.

5.2 Clasificación por forma

Sería muy interesante intentar extraer algunas líneas de margen, y esto se puede obtener aplicando una función que compara píxeles adyacentes (en todas las direcciones) y devuelve un valor de cuán diferentes son esos dos píxeles.

Al hacerlo puedemos analizar en cada dirección un perfil de intensidad donde el pixel de borde que me interesa será el pixel que actúa como máximo local. Aplicando esta función en cualquier dirección puedo obtener todos los máximos locales y, desde aquí, unir los puntos adyacentes obteniendo líneas de perfil como en la fig 8.

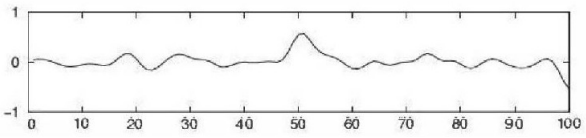

Figure 8: Una derivada de una versión aplanada de la función de intensidad uni-dimensional: el candidato es 50

References

[1] Artificial Intelligence: A Modern Approach *Fourth edition, by Stuart Russell and Peter Norvig* 2020.

[2] https://www.robots.ox.ac.uk/~vgg/data/flowers/17/index.html *17 Category Flower Dataset*

[3] Genetic k-means algorithm. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics. *Krishna, K.; Murty, M. N. (1999).*