Clustering - conjunto de colores

Luca Manzi
Ingeniería Informática
ETSII
Sevilla

01-02-2021

1 Objetivo

Utiliza algún algoritmo de Clustering para rebajar el número de colores que se usa en una imagen. Para ello:

- Carga una imagen en los patches haciendo uso de la función de importación adecuada.
- Aplica el algoritmo de Clustering al conjunto de colores, determinando a priori el número de colores finales que quieres usar (puedes hacer que sea una decisión del usuario por medio del interfaz).
- Muestra la imagen resultante con los colores reasignados según el cluster al que pertenecen.
- ¿Se te ocurre alguna forma de restringir la paleta de colores original (no solo el número de colores)?

2 Desarrollo

2.1 Importar una imagen

El concepto clave que se debe solucionar inicialmente es seleccionar la función de importación adecuada; pero la pregunta que necesitamos ponernos es: ¿Cómo se puedan codificar los colores de una imagen?

Hay varios métodos, el mas conocidos es:

- **RGB { #, #, #}** : el espacio de color RGB o sistema de color RGB, construye todos los colores a partir de la combinación de los colores rojo, verde y azul. Normalmente usan 8 bits cada uno, que tienen valores enteros de 0 a 255. Esto hace que $256 \times 256 \times 256 = 16777216$ colores posibles.

Sin embargo, esta codificación no es tan eficiente para mi propósito, por que lo único que tengo que hacer es calcular la diferencia entre colores. Se puede usar la distancia euclidiana, pero solo es eficiente matemáticamente, no para el ojo humano. La formula es :

$$\sqrt{(R_2 - R_1)^2 + (G_2 - G_1)^2 + (B_2 - B_1)^2}$$

Es muy simple, pero lamentablemente el ojo humano no es tan sensible a las diferencias entre colores sino al brillo y otros factores.

Por lo tanto, como no tenemos mejoras notables con el uso de este sistema, para hacer aún más inmediato el cálculo de la distancia, podemos utilizar el sistema de color nativo de netlogo donde cada color puede ser un número en el rango de 0 a 140, con la excepción del propio 140.

Para esto la función que vamos a utilizar es: **import-pcolors filename** :

> Reads an image file, scales it to the same dimensions as the patch grid while maintaining the original aspect ratio of the image, and transfers the resulting pixel colors to the patches. The image is centered in the patch grid. The resulting patch colors may be distorted, since the NetLogo color space does not include all possible colors. (See the Color section of the Programming Guide.)

import-pcolors may be slow for some images, particularly when you have many patches and a large image with many different colors.

Since import-pcolors sets the pcolor of patches, agents can sense the image. This is useful if agents need to analyze, process, or otherwise interact with the image. If you want to simply display a static backdrop, without color distortion, see import-drawing.

The following image file formats are supported: BMP, JPG, GIF, and PNG. If the image format supports transparency (alpha), then all fully transparent pixels will be ignored. (Partially transparent pixels will be treated as opaque.)"

Todo esto es perfecto para trabajar con los píxeles de la imagen, lo que tenemos que hacer, también se puede cambiar la cuadrícula para tener diferentes dimensiones en las que probar el algoritmo.

2.2 Setup - Configuración

Ahora necesito trabajar con la imagen que acabo de importar

Normalmente podríamos haber usado una función : **sort patches** para convertir la cuadrícula de patches en una lista, pero ya puedo pensar en ordenar por color:

```
to gen-state
  let b map [x ->
    (list
      [pxcor] of x
      [pycor] of x
      [pcolor] of x
  )] (sort-on [(- pcolor)] patches)
  set state b
end
```

De este modo cada elemento de la lista **state** será como:

[x y color]

El resto de la configuración es autoexplicativa.

```
to start
  ask patches [set pcolor 0]
  import-pcolors "image.jpg"
  gen-state
  set maxpix length state
  set-colorz
end
```

```
to set-colorz ;;counts color diff: colordiff + 1 = total number of colors
  let l1 bl state
  let l2 bf state
  let diff (map [ [x y] -> last x - last y ] l1 l2)
  let quante 0
  foreach diff [ x ->
    if (x != 0)
    [set quante quante + 1]
  ]
  set colorz quante + 1
end
```

2.3 Distancia

Ahora con una lista de patches ordenadas por color, tengo que obtener la distancia entre ellas: el sistema que voy a usar es superponer una lista sobre si misma *shiftada* de una posición y así producir una lista de diferencias (que tendrá un elemento menos de la lista original)

$$\begin{bmatrix}138 & 137 & 135 & 135 & 134 & \cdots\end{bmatrix} -$$
$$\begin{bmatrix}137 & 135 & 135 & 134 & 134 & \cdots\end{bmatrix} =$$
$$\begin{bmatrix}1 & 2 & 0 & 1 & 1 & \cdots\end{bmatrix}$$

Ahora puedo usar un algoritmo de clustering para agrupar los datos, en este caso usaré un algoritmo de tipo AGNES.

```
to ag2 [l]
  let diff difflist l
  let p get-posmin diff
  if (p >= 0 )[ ;; cause -1 is the default value
    let nl how-many-equals-left diff p
    let nr how-many-equals-right diff p
    let newcolor2 last item (p + 1) l
    let newcolor1 last item p l
    let newcolor -1
    let i 0
    ifelse (nr <= nl)[ ;; busco el lado con menos valores iguales
     set newcolor last (item p  l)
     while [i <= nr][
      set p (p + 1)
        if ((p + i) < maxpix) [
        ;; this is because on the right side there is everytime a space more
        ;;(cause its a substraction of lists),
        ;;and i have to check if i am over the boundary
          ask patch (first item (p + i) l) (first bf item (p + i) l)
                [ set pcolor newcolor ]
        ]
        set i i + 1
      ]
    ]
    [
      set newcolor last item (p + 1) l
      while [i <= nl][
      ask patch (first item (p - i) l) (first bf item (p - i) l)
        [ set pcolor newcolor ]
      set i i + 1
      ]
    ]
  ]
end
```

Explico brevemente:

- **get-posmin diff** : devuelve la posición del minimo (mayor que 0) en la lista de diferencias.
- **how-many-equals-left y right** : devuelve el numero de ceros cerca de la posición del minimo.
- **newcolor** : selecciono y asigno a newcolor el color del lado con menos valores iguales

Figure 1: Full-resolution sobre Netlogo 1161 colores y 81 KB

Figure 2: 512 colores y 61 KB

Figure 3: 256 colores y 50 KB

Figure 4: 128 colores y 42 KB

Figure 5: 62 colores y 36 KB

3 Restringir la paleta de colores original

Para reducir el tamaño, se puede asignar cada píxel a un color de una tabla de riferimento. Si se desea aplicar una paleta preestablecida, necesitamos calcular la distancia menor desde el color original de cada píxel a uno de los colores de la paleta, que actuará como un centroide.

Otra idea podría ser darle una importancia a cada píxel o a cada grupo de píxeles: es decir cuánto aporta visualmente a la imagen, si es muy diferente en colores a lo que hay alrededor, etc. y con estos parámetros ajustar nuestros algoritmos.

De otra manera, se podría dividir grupos de píxeles algebraicamente, por ejemplo con una descomposición como la de taylor, de una manera similar a como lo hace con el algoritmo jpeg, incluso si usa otros parámetros y no solo el color y una descomposición basada en la función coseno a partir de algunas tablas.

Así para que se pueda reconstruir algo muy similar al grupo con un tamaño menor de información.