Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-5438 Inteligencia Artificial II
Trimestre Septiembre - Diciembre 2023

Proyecto 3

Estudiantes:

Figueira, Adelina

Bandez, Jesús

Carnets:

15-10484

17-10046

Profesor

Carlos Infante

8 de diciembre de 2023

Detalles de la implementación

Como el algoritmo de kmeans se trata acerca de dividir un conjunto de datos en varios clusters disjuntos, se implementó el cluster como una clase que tiene como variables el centro del cluster y los puntos que se encuentran cercanos a dicho cluster como una lista de numpy arrays, además se incluyeron los métodos addpoint y un método para hacer override del string y poder visualizar cada centroide desde la consola junto a los puntos que lo conforman.
El algoritmo kmeans se implementó dentro de una clase de nombre K_means. Esta clase posee las variables n, X y clusters que representan la cantidad de clusters, el archivo que se desea analizar y una lista de n 0s en donde se albergarán los clusters, respectivamente. La clase K_means cuenta con varios métodos:

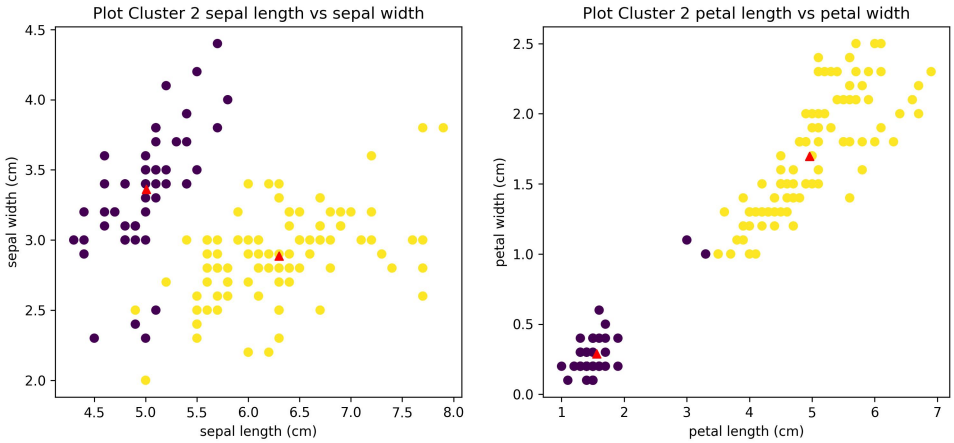
- initialize: Se encarga de inicializar la lista de clusters con un centro igual a la fila del centro de la matriz del archivo (la que se encuentra junto en la posición M/2 donde M es el número de filas de la matriz) que se desea analizar y una lista de puntos vacía.
- distance_x_y : Este método calcula la distancia euclidiana entre dos puntos, siendo los puntos np.arrays.
- assign : Este método se encarga de iterar sobre cada uno de los ejemplos del archivo y calcular la distancia entre el ejemplo y los clusters, toma la distancia mínima y para el cluster que tiene distancia mínima, le agrega el ejemplo como un punto.
- updating: Este método actualiza los valores de los centroides y los puntos para cada uno de los clusters. Los puntos se inicializan como una lista vacía y los centroides se calculan basándose en la mediana de todos los puntos que tenía el centroide hasta el momento.
- fit : Este método ejecuta el algoritmo de K means ya que se inicializan los clusters, establece un minimo local e itera una cantidad X de veces aplicando los métodos de assign y updating para conseguir disminuir las distancias entre los clusters y cada uno de los puntos.
- predict : Una vez se ha entrenado el algoritmo con el archivo original, este método se usa para buscar el cluster de distancia mínima para cada uno de los ejemplos de una matriz

X y luego se calcula el mínimo de todas las distancias para cada cluster, consiguiendo así el número del cluster al que pertenece el ejemplo X_i, consiguiendo así la clase a la que pertenece.

También se implementó un método plot que toma los valores de predicción de K_means y genera varios gráficos que comparan las variables sepal length vs sepal width y petal length vs petal width que se encuentran en el archivo iris.csv.

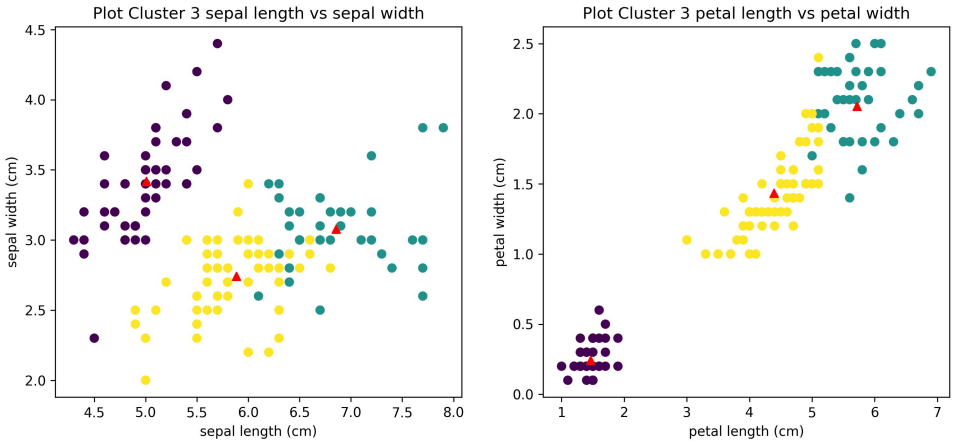
Análisis de resultados para iris.csv

Para k=2



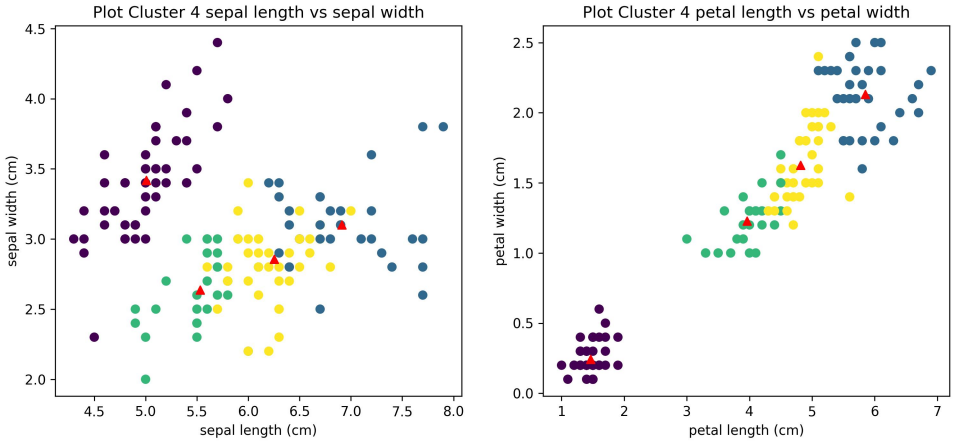
Es posible ver en el gráfico que para el caso de 2 clusters, se logra dividir de manera bastante apropiada los datos de iris.csv.

Para k=3



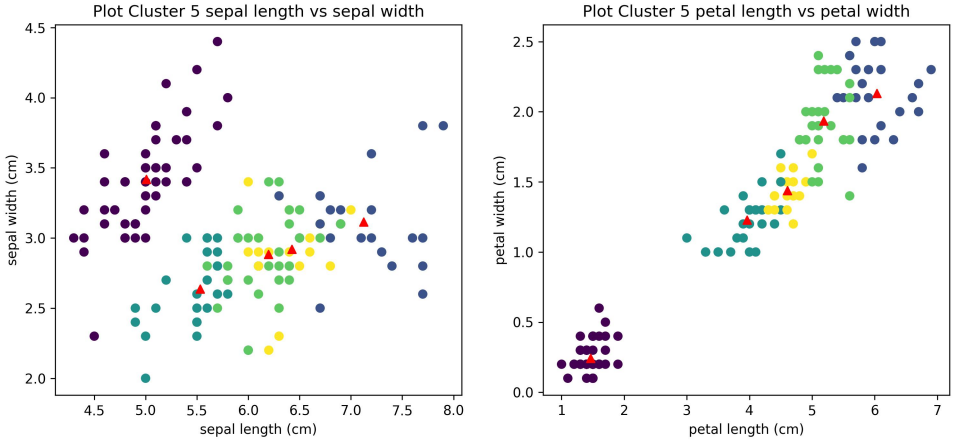
Para el caso de 3 clusters, la división entre los conjuntos es bastante buena para ambos gráficos.

Para k=4



Para el caso de 4 clusters, existe una separación bastante pequeña para los clusters amarillo y verde del gráfico petal length vs petal width lo que nos puede llevar a dudar de la veracidad de la clasificación realizada al agregar más de 3 clusters, ya que nuestro archivo solo contiene 3 clases.

Para k=5



Para el caso de 5 clusters, ocurre lo mismo que para el caso de 4 clusters, con 2 de los clusters ubicados muy cerca uno del otro.

Por esto el mejor k que podemos considerar es 2 o 3, pero como nuestro archivo iris.csv cuenta con 3 clases, el mejor k para la separación es clases es el k=3.

Para el k=2 se consiguieron los siguientes datos de predicción:

```
setosa =  [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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
↪  0, 0, 0]
versicolor =  [1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
↪  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
↪  1, 1, 0, 1]
virginica =  [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
↪  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
↪  1, 1, 1, 1]
```

En donde setosa, versicolor y virginica son las predicciones de cada una de los ejemplos que pertenecen a dichas clases. Se pueden clasificar únicamente en 2, por lo que no es suficiente para el ejemplo de iris.csv.

En cambio si consideramos k=3, podemos ver que:

```
setosa =  [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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
↪  0, 0, 0]
versicolor =  [1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
↪  2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
↪  2, 2, 2, 2]
virginica =  [1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 1,
↪  2, 1, 2, 1, 1, 2, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1,
↪  2, 1, 1, 2]
```

Se clasifica de manera correcta a todos los elementos de iris-setosa como una única clase, mientras que para la clase iris-versicolor hay un 0.94 de exactitud y para la clase iris-virginica hay un 0.72 de exactitud.

Dados estos resultados es posible afirmar que la separación en clusters tiene una relación con las clases originales siempre que se elija un k correcto para poder clasificar la información de manera adecuada.

Similar al algoritmo de propagación hacia atrás, que obtuvo un mínimo de exactitud de 0.66, K means es capaz de predecir con una exactitud de al menos 0.72 para todas las clases. El

resultado observado para iris-virginica para K means que obtiene una menor exactitud que otras clases puede deberse a que en el caso del algoritmo de propagación hacia atrás, esta clase requería de una mayor cantidad de iteraciones al ejecutar el algoritmo con una única capa oculta y se comportaba de manera diferente al resto de las clases, dando una gran cantidad de falsos positivos y negativos al momento de operar con un alpha diferente de 0.001.

Generación de imagenes con paleta de k colores

Una imagen es una matriz de pixeles de tamaño mxn . En el formato RGB, un pixel tiene tres valores. Estos valores corresponden a la intersidad de Rojo, Verde y Azul. Se puede decir que un pixel no es más que un vector de tres elementos. Ahora, cada uno de estos elementos debe estar comprendido en el intervalo $[0, 255]$. Por tanto, es posible representar un total de 256^3 colores en el formato RGB. Cada uno de estos colores puede ser visto como una clase. Además, se pueden agrupar a estos colores según determinadas reglas o según su cercanía. Definiendo la cercanía de un color a otro como una cierta diferencia entre sus valores representados en RGB. Es posible conseguir una agrupación de colores utilizando el algoritmo de k-means. Esto se logra entrenando un modelo con los pixeles de una imagen, asignando cada pixel a un centroide y asignando un color a dicho centroide. Luego, se intercambian todos los pixeles de la imagen original con el color al que corresponde el centroide al que se le asigna al pixel. A continuación, se mostrarán varias imágenes que fueron procesadas con el algoritmo de k-means para visualizar la asignación de clusteres.

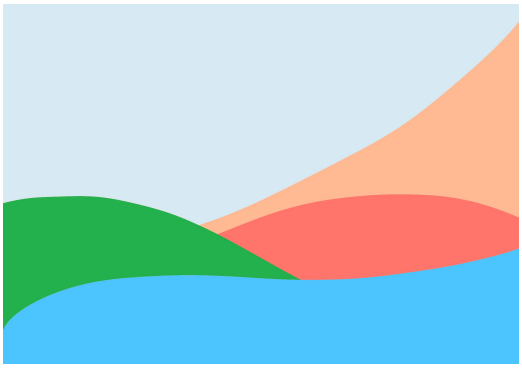
Asignación de paleta de k-colores

Para crear la paleta de k colores, primero se ejecuta el algoritmo para conseguir la asignación de clústeres. Luego, se busca la posición de la primera ocurrencia de una asignación del cluster i. Con esta posición, se busca el pixel que está en ella pero en la imagen original. El color de dicho pixel es asignado como el color del clúster.

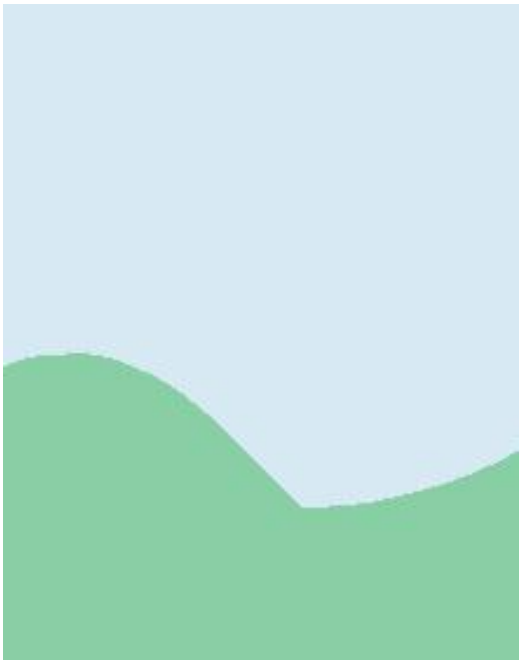
Se realizaron 5 experimentos sobre las imágenes, de forma que estas son analizadas con cinco valores de $k = (2, 4, 8, 16, 32)$

Colores sólidos

Para la primera imagen, se utilizó una que sólo contiene colores sólidos. Y, para ser exactos, tiene 5 cólores. Esto es, la imagen no posee cambios de color por iluminación, degradados ni saturación:

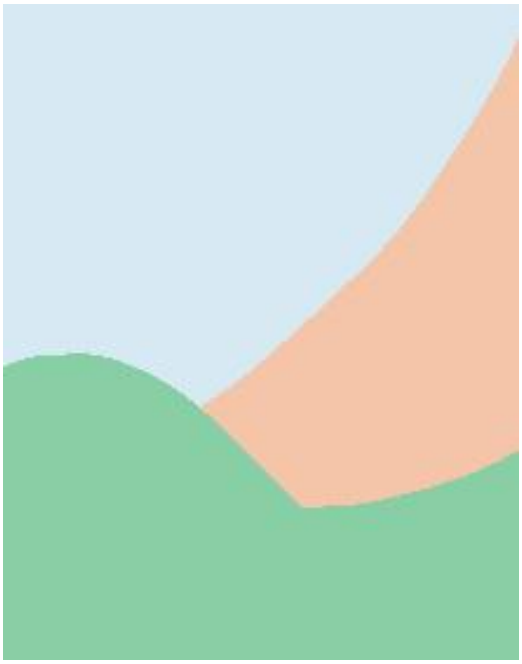


Utilizando una paleta de $k = 2$ colores se obtiene:

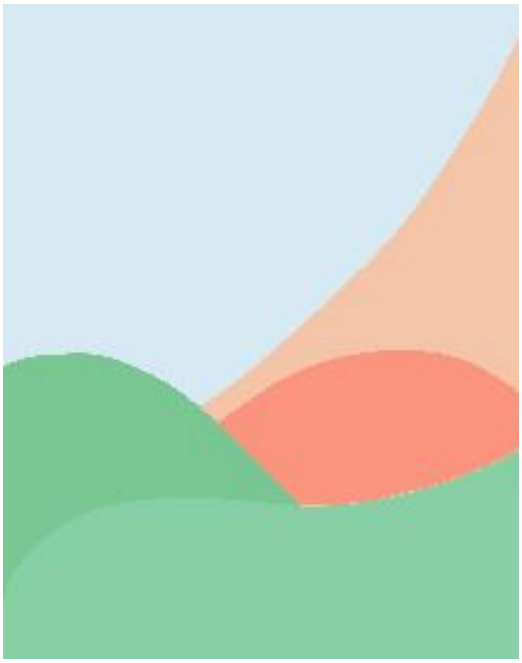


Note que la imagen se divide en dos partes. Una azúl y verde. Esto indica que el algoritmo de clusteres reconoce dos colores principales para dividirla. Esto tiene sentido, pues, en la imagen original, la parte inferior cuenta está compuesta por colores con mucha intensidad a diferencia de los colores de la parte superior.

Veamos que cólores se consiguen con $k = 4$

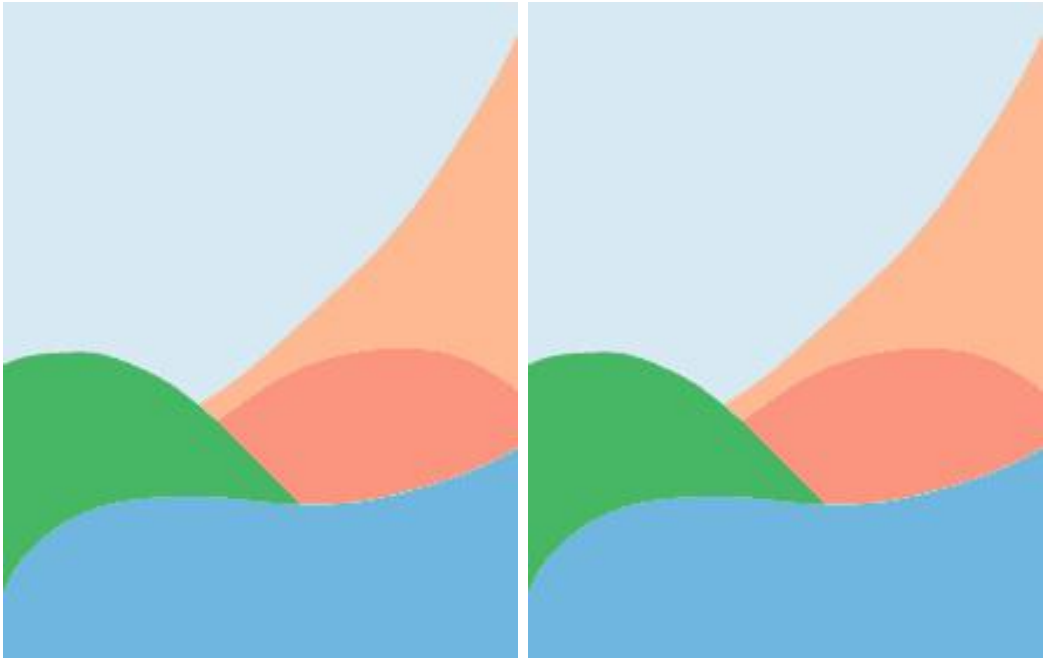


En este caso, se ha creado la diferencia entre el color azul y el naranja. Sin embargo, si k=4, ¿donde está el color que falta? Aparentemente, el algoritmo no consiguió separar lo suficiente los colores para poder formar un cuarto clúster. Veamos si esto sigue siendo una problemática para las siguientes imágenes.



Esta vez el algoritmo logró separar correctamente todos los colores de la imagen original. Aunque no restauró los colores originales, sí es notable la separación entre ellos.

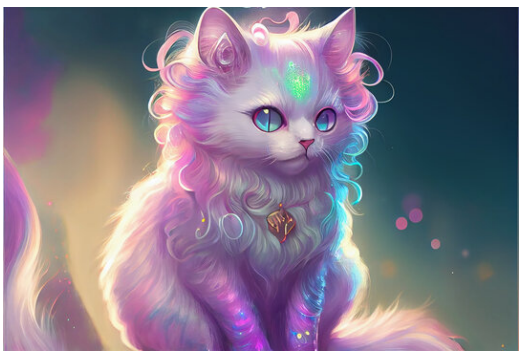
Al no existir más de 5 colores en la imagen, no tiene sentido seguir ejecutando el algoritmo en búsqueda de nuevas agrupaciones. De igual forma, veamos qué imagenes se obtienen la hacerlo.



En las imágenes no hay cambios en la agrupación de los colores. Despues de todo, la imagen original sólo tiene 5 colores posibles para agrupar. Aunque cabe mencionar que aumentar los clusteres mejoró la asignación de colores. Esto es simple casualidad.

Alta iluminación y gradiente de colores

Las segunda imagen utilizada introduce los cambios de colores debido a la iluminación. Además, utiliza colores muy similares que varían con pocas tonalidades:



Ejecutando el algoritmo con 2 clústeres se obtiene:

En este caso, el algoritmo logra separar dos grupos: Colores oscuros y colores claros. Note que la imagen original está compuesta por colores oscuros a la derecha y claros a la izquierda. La version con dos colores está conformado por un contorno oscuro y contorno claro. Es fácil divisar la diferencia entre el fondo de la imagen y la figura del gato. De igual forma, es sorprendente que con sólo dos colores pueda mostrarse de forma clara los ojos de este. Puede decirse que la iluminación juega un papel importante en la separación de los grupos. Agregando dos colores nuevos, se consigue la siguiente imagen:

Con esta versión, empieza a verse la importancia de la iluminación en la imagen. En esta, se muestran cuatro colores, dos oscuros y dos claros: Una tonalidad cercana al morado oscuro, que cubre el fondo de la imagen; azul oscuro tendiendo a gris, aparentemente utilizada para destacar la diferencia entre los tonos claros y oscuros; El gris, que marca una frontera para los colores claros y finalmente el blanco, que representa los colores más claros. Cuando se usan ocho clústeres, esta diferencia entre la iluminación es más notable

En esta imagen, la forma del animal está más definida, pues, son claramente visibles las orejas, los pelos y parte de la cola. Incluso, pueden verse los detalles de la sombras en su pelaje. Los ojos está mucho más detallados y pueden verse sus pupilas. De igua forma, note que no es posible distinguir la forma de la cola. Aumentemos los clústeres a un total de 16 para ver más cambios.

Puede verse que esta nueva paleta de colores intenta reproducir los detalles de la imagen con mayor fidelidad. Lamentablemente, la imagen original posee demasiados cambios de iluminación y muchos colores con degradación. Esto hace que el resultado parezca algo fuera de lugar con respecto a la coloración. Sin embargo, sí es posible afirmar que el algoritmo es capaz de separar los niveles de iluminación. Veamos la imagen con 32 clústeres:

No parece existir mucha diferencia con respecto a la imagen anterior. Sin embargo, pueden

notarse cambios minimos entre ellas. Por ejemplo, la coloración de las orejas y los ojos del gato. En esencia, la imagen con $k = 32$ clústeres tiene más detalles que la de $k = 16$ clúster.

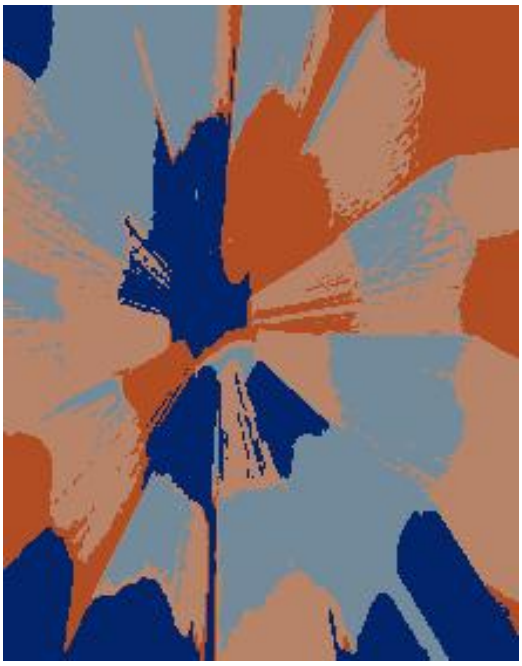
Colores sólidos pero con detalles y poca iluminación

Utilizaremos una última imagen para ejecutar el algoritmo. En este caso se tienen una serie de colores presumiblemente sólidos, parecidos a la primera imagen utilizada para los experimentos. Pero, esta vez se incluyen texturas con detalles rugosos y sutiles cambios de iluminación.

Ejecutar el algoritmo con sólo 2 clústeres genera la siguiente imagen:

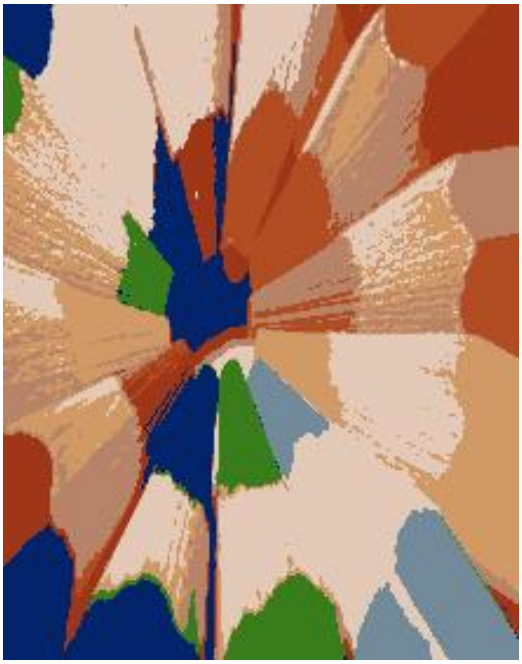
En la imagen original se puede ver que domina el color marrón por la madera de los crayones. La imagen con sólo 2 cluster demuestra esto al separar con un color claro a las secciones con

el color de la madera y con el color oscuro a las secciones que no son ella. Agreguemos los dos nuevos colores para ver la diferencia.



En esta imagen puede verse que se repite lo mismo que la imagen del gato. El algoritmo intenta conseguir una diferencia entre la iluminación de la imagen. Sin embargo, esta vez los colores también juegan un papel importante en la separación. Note que los crayones son separados en dos colores: azul y naranja. Los colores que fueron reemplazadas en la imagen original corresponden al color azul y rojo. Podría decirse que en esta imagen el algoritmo agrupa a dos colores opuestos y a dos colores para destacar la iluminación.

Agregar cuatro colores más a la paleta retorna el siguiente resultado:



Al tener ocho colores posibles, el algoritmo logra agrupar una subpaleta de colores decente para poder mostrar la diferencia de iluminación en la imagen. Además, es capaz de diferenciar entre los colores: Marrón claro, Verde, Azul y Cyan. Cabe destacar que en este punto empiezan a tomar importancia los detalles del relieve producido por la madera en los crayones. Sin embargo, la paleta no es lo suficientemente amplia para mostrarlos con claridad.

Duplicar la cantidad de colores disponibles resulta en la siguiente imagen:

El algoritmo ya es capaz de agrupar los colores de forma más discreta. Puede verse la diferencia entre el rojo y el naranja. Así como la diferencia entre el mismo naranja y amarillo. Sin embargo, aún no los reconoce a todos. No puede verse la diferencia entre el crayón violeta, el azul oscuro y el verde oscuro.

Aumentando la paleta de colores a $k = 32$ se obtiene:

Ahora es posible notar el crayón rosa. Pero no hay una diferencia importante con respecto a $k = 16$. Se hará un último experimento, aumentando el número máximo de iteraciones en el algorito de convergencia y usando un $k = 64$. El resultado de ello es la siguiente imagen:

El experimento retorna un buen resultado. Veamos que el color verde oscuro puede diferenciarse del verde claro. Además, el rosa obtiene un coloración distinta que lo remarca más como tal. Es interesante el caso del violeta. Note que está pintado con un color muy parecido al azul mezclado con blanco. Además, note que dicho color no puede verse en ninguno de los otros colores. Esto tiene una explicación, es posible que algoritmo realmente fue capaz de obtener un cluster para el violeta. Lamentablemente, el color que se asigna para intercambiar el clúster termina siendo esa tonalidad azul. Por tanto, se puede afirmar que el algoritmo es capaz de diferenciar todos los colores de los crayones presentes en la imagen.

Referencias

- Luke, T. (2022, Abril 10). Create a K-Means Clustering Algorithm from Scratch in Python. https://towardsdatascience.com/create-your-own-k-means-clustering-algorithm-~:text=Introduction,variables%20and%20no%20dependent%20variables.
- Russell, S. J., y Norvig, P. (2020). Artificial intelligence: A Modern Approach.