Cross Stitching Images with Unsupervised Learning: Starting with K-Means Algorithm

Samantha Wong

November 2, 2020

Creating a Cross-Stitch Pattern

Introductiion

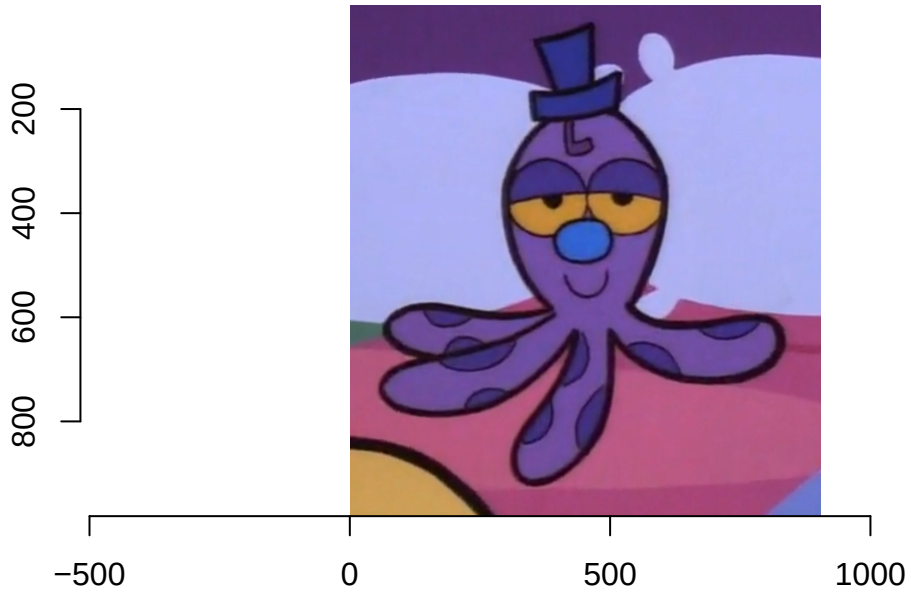
This vignette explains how to use certain functions formed from packages in statistical software R to perform k-means clustering on images in R based on their corresponding color to dmc floss colors. Performing k-means clustering on the colors in images will guide us to create a cross-stitch pattern by simplifying the image.

K-means clustering is one of the simplest unsupervised machine learning algorithms. Unsupervised learning's goals are to make inferences on the datasat using only the input without looking into known outputs. K-means clustering groups data points that are similar together to find patterns about the data. We will start off by defining a number of centers of clusters. By reducing the cluster sum of squares, data points will join certain clusters. These clusters will inform us about the patterns in the data to help categorize them. In this case, we will be clustering the different colors found in an image and matching them with dmc floss colors. We will first run the k-means algorithm with a chosen value of k(number of centers), then plot a scree plot to find the best value of k. Lastly we will plot the clustered data to create the cross-stitch of the image.

In order to run the functions listed, please install and load packages:

1. imager
2. tidyverse
3. tidymodels
4. sp
5. scales
6. cowplot
7. dmc

In this example, we will be loading an image of Bubbles' stuffed octupus doll Octi from the Powerpuff Girls to perform k-means clustering.



This first function *process_image* takes in two inputs: 'image_file_name' (the file of the image in .png or .jpeg format we want to cluster) and 'k_list' (number of centers we want in the clustering). The ouput produces four separate tibbles, one with the original kmeans output, the kmeans output that was has mapping and was glanced, a tidy version of the kmeans output with the corresponding RGB and DMC color for each cluster center and an augmented version of the kmeans output that was glanced.

```
#Function process_image

process_image <- function(image_file, k_list){

  tidy_dat <- as.data.frame(image_file, wide = "c") %>%
    rename(R = c.1, G = c.2, B = c.3)
  dat <- select(tidy_dat, c(-x,-y))

  #Mapping out the klustering to create screeplots
  kclusts <-
  tibble(k = c(2:(k_list + 2))) %>%
    mutate(
      kclust = map(k, ~kmeans(x = dat, centers = .x, nstart = 4)), glanced = map(kclust, glance)
  )
  #Produce the tidy clustering information and adding dmc hex color
  kclust <-kmeans(select(tidy_dat, -x, -y), centers = k_list, nstart = 20)
  centers <- tidy(kclust)%>%
    mutate(rgb_col = rgb(R,G,B))


  #adding the corresponding DMC floss color
```

```
  dmc_col <- tibble(rgb_col = "", dmc_code = "")
  for (i in centers$rgb_col){
    dmc_col <- add_row(dmc_col, rgb_col = i, dmc_code = dmc(i)$hex)
  }
  dmc_col <- print.data.frame(dmc_col, quote = FALSE)[-1,]
  dmc_centers <- left_join(centers, dmc_col, by = "rgb_col")

  #Augmenting kclusts
  tidy_dat1 <- augment(kclust, tidy_dat) %>% rename(cluster = .cluster)
  output <- list(kclusts, kclust, dmc_centers, tidy_dat1)
return(output)
}

#Putting through the image of Octi and setting number of centers to 8
octi_result <- process_image(octi_im, 8)
```

```
##   rgb_col dmc_code
## 1
## 2 #8F6C39  #985E33
## 3 #1E0B1C  #1E1108
## 4 #823D5C  #803A6B
## 5 #3B2554  #213063
## 6 #2C1A38  #1B2853
## 7 #513A72  #464563
## 8 #7F81B2  #7B8EAB
## 9 #435191  #466A8E
```

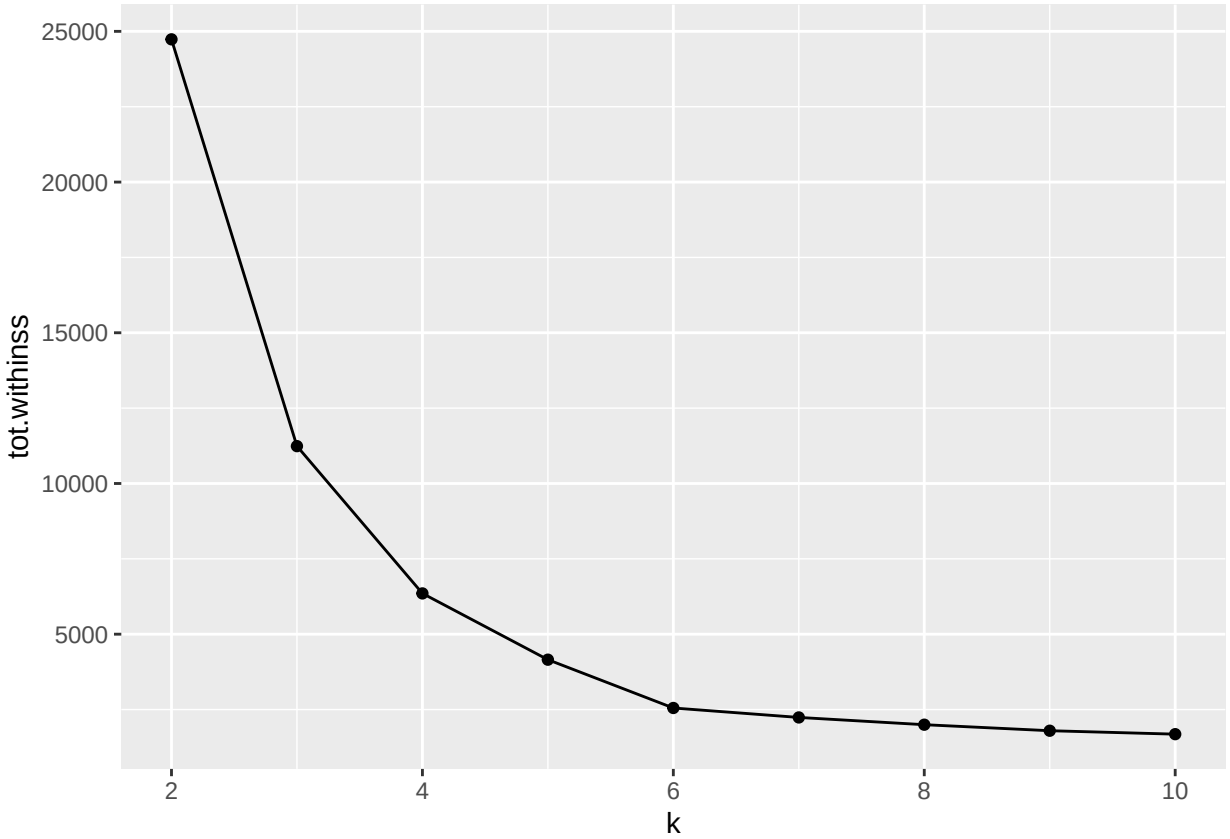
Producing Scree Plots

This next function produces a scree plot for the number of clusters we may be interested in looking at. A scree plot is a great tool in multivariate statistical analysis since it plots a line of the eigenvalues in an analysis. When looking at a scree plot to choose an appropriate k value, we would like to select a value of k that sees the eigenvalues drop in size. The input will be the results from running the first function and will specifically call upon the clustered information that was put through the *glace()* function. I have saved the return as 'cluster_info' and I can call upon either the kclusts tibble or the tidy tibble with the DMC color in the list from the output.

```
scree_plot <- function(cluster_information){

  kclusts <- as.data.frame(cluster_information[1])
  clusterings <- kclusts %>%
    unnest(cols = c(glanced))
  #Plotting the scree plot based on the output of the first function
  scree_plot1 <- ggplot(clusterings, aes(k, tot.withinss)) +
  geom_line() +
  geom_point()
  return(scree_plot1)
}


scree_plot(octi_result)
```



In the scree plot produced, we can visualize where the best selection of k would be. Based on this plot, the best selection of appears to be around 7, however, we can take advantage of the fact that we are clustering colors and use a visualization to directly see the best number of clusters.

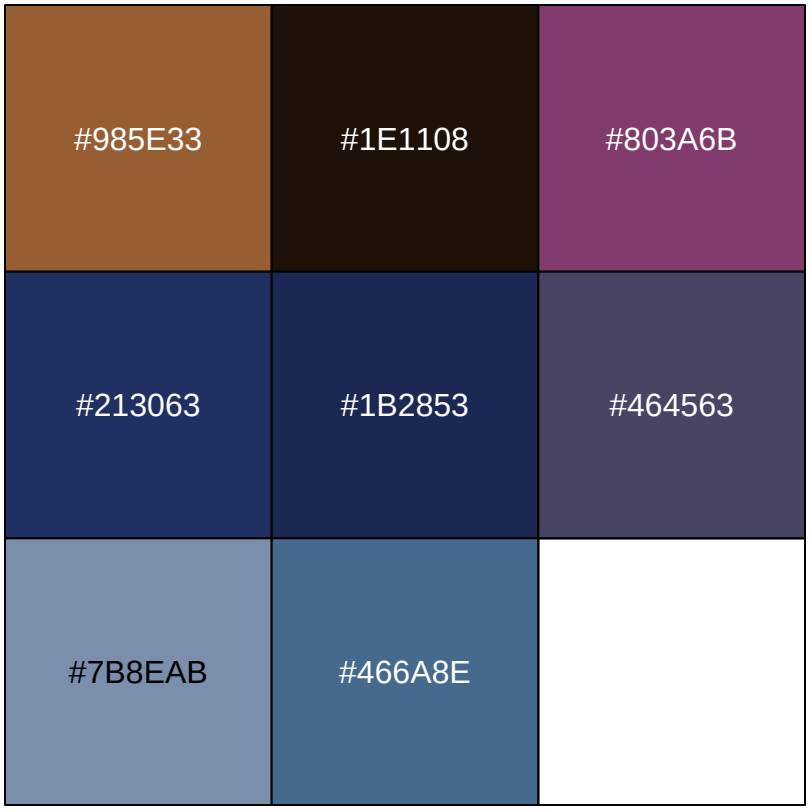
Colour Strips

In ths *colour_strips* function, we will be producing strips with the DMC floss color closest to the cluster center color. We will be using the *dmc()* function from the *DMC* package to find the closest DMC floss color to the cluster centre color and producing a colour strip. Our input will also be the ouptut from our first function, *process_image* since it has all our clustering information.

```
#Plotting the color blocks based on output from the first function
colour_strips <- function(cluster_information){
  dmc_centers <- as.data.frame(cluster_information[3])
  color_plot <- show_col(dmc_centers$rgb_col)
  dmc_plot <- show_col(dmc_centers$dmc_code)
  output1 <- list(color_plot,dmc_plot)
  return(output1)
}

strips <- colour_strips(octi_result)
```


#8F6C39
#1E0B1C
#823D5C
#3B2554
#2C1A38
#513A72
#7F81B2
#435191



These show the colors that are the cluser centers. The second color square shows the corresponding DMC colors. Since we are clustering colors, we can look at which cluster center colors are similar to each other and try to reduce the number of k. Those two shades of reddish purple look suspiciously similar. Based on this information we could reduce the number of k clusters by one since we do not need two very similar looking reds(it looks of a magenta in the DMC colors). But we are fortunate to have a visual of the actual picture we are clustering. Based on the octi.png, the shading on the floor is causing two slightly different shades of red/purple. I like the shade variation, so I am going to keep the two colors in but this is just my personal preference. Depending on what you are performing k-means analysis on, similarities in cluster centers could mean different things so always consider the context before decreasing amount of cluster centers.

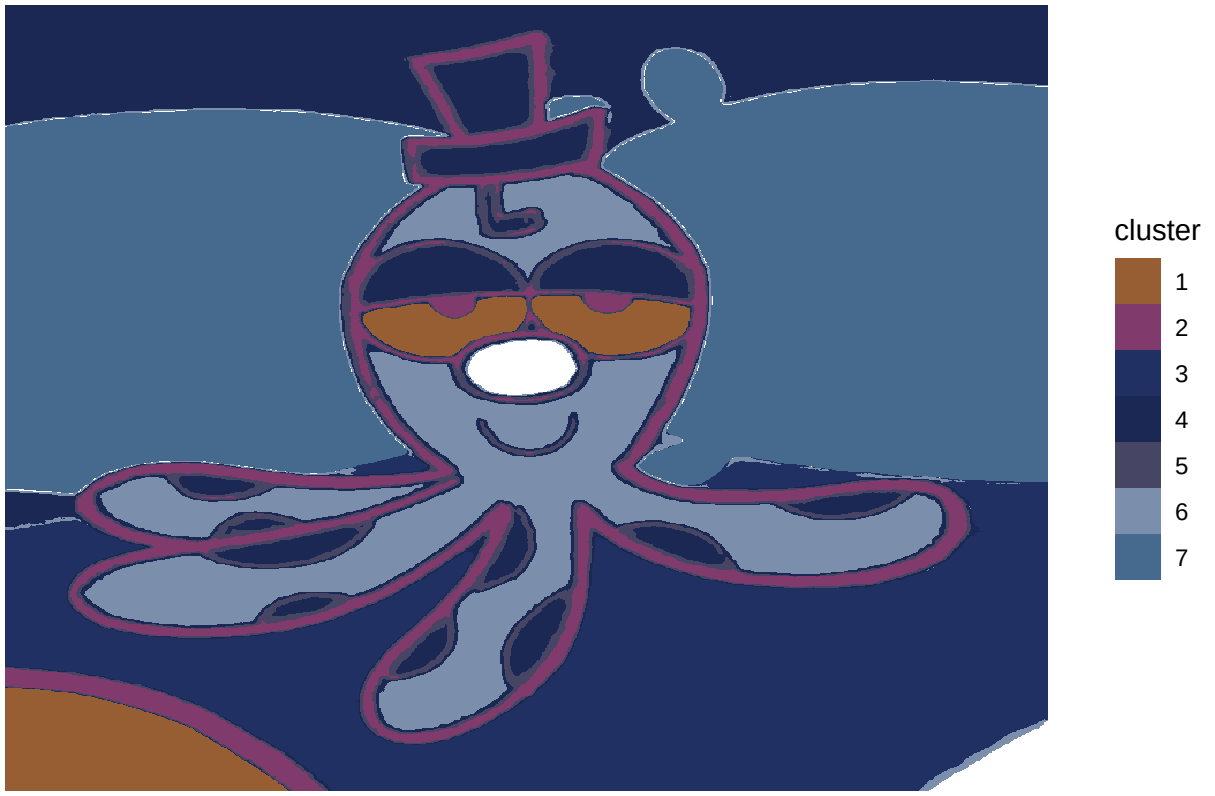
Plotting Patterns

We will now be plotting the clustered picture, or our cross-stitch pattern! Our input is the output of process_image, the chosen cluster size, the total number of possible stitches in the horizontal direction, the choice to print the image in black and white or color and the color of the background that is not being stitched into the pattern. The default for black_white is FALSE, meaning it will not print in back and white and the default for background_colour is NULL, meaning there is no background color to omit in the cross-stitching.

```
#State the Function
make_pattern <- function(cluster_information, k, x_size, black_white = FALSE, background_colour = NULL)
  #Selecting amount of clusters to be plotted
  plot_data<- as.data.frame(cluster_information[4]) %>%
    filter(as.numeric(cluster) <= as.integer(k))
  #Change for when a background color is stated
```

```
  ifelse(is.null(background_colour) == FALSE,
         dmc_info <- as.data.frame(cluster_information[3]) %>%
           filter(dmc_code != background_colour),
         dmc_info <- as.data.frame(cluster_information[3]))
  ifelse(black_white == TRUE,
          cross_stitch <-
           ggplot(plot_data, aes(x=x, y=y, fill = cluster)) +
           geom_tile() +
           scale_y_reverse() + theme_void() + scale_fill_grey(),
         cross_stitch <-
           ggplot(plot_data, aes(x=x, y=y, fill = cluster)) +
           geom_tile() +
           scale_discrete_manual(aesthetics = "fill", values = dmc_info$dmc_code) + scale_y_reverse() +

  return(cross_stitch)
}
## Only 7 clusters with a background color ommitted
cross_stitch <- make_pattern(octi_result, 7, background_colour = "#1E1108")
cross_stitch
```




```
## Nothing altered
cross_stitch1 <- make_pattern(octi_result, 8, background_colour = NULL)
cross_stitch1
```


```
#Black and White version!
cross_stitch2 <- make_pattern(octi_result, 8, background_colour = NULL, black_white = TRUE)
cross_stitch2
```

I have plotted out different cros-stitchings of Octi by inputting different parameters. Since this function does not compute any k-means, you should reproduce results with different number of clusters in the first function to find different clusters nd cluster centers. You can reproduce this process with any image, however the higher resolution it is, the more difficult and longer it takes to run this algorithm. Octi was 980 X 980 pixels and I still got a bit impatient running the kmeans algorithm over and over again. If you wish to perform kmeans clustering on images with high resolutions, you can use the change resolution function to decrease the amoung ot pixels and thus simplify your calculation of kmeans.