KIT306/606 Tutorial 8

Student ID	Name

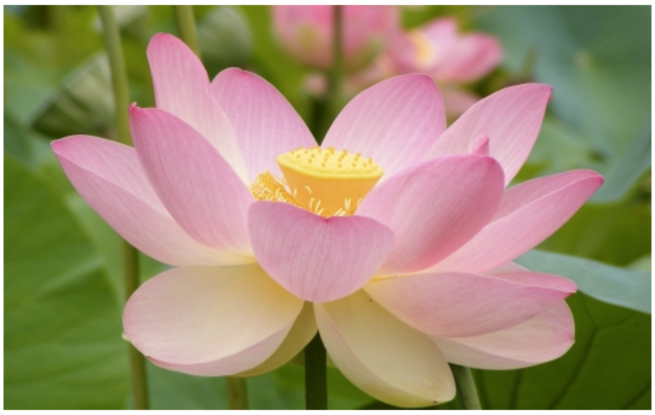
The following tutorial work should be completed by tutorial 9 (week10).

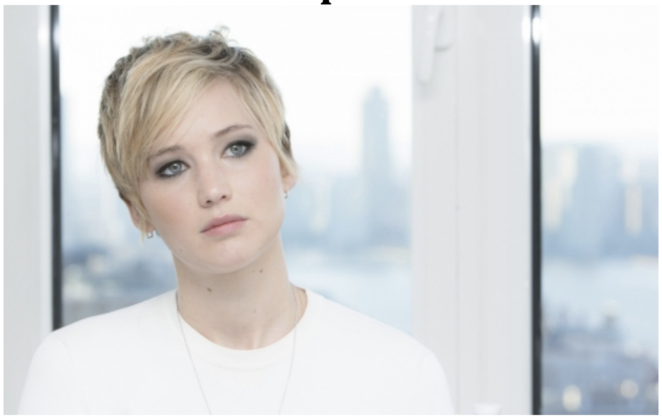
Last week, we have learned the practical work in applying clustering techniques, including k-means clustering and hierarchical clustering.

This tutorial will guide you how to apply clustering technique in the real-world problem. In recent years there has been a growing interest in developing effective methods for content-based image retrieval (CBIR). Image clustering is one of the most popular approaches for high-level description of image content. The goal is to find a mapping of the archive images into clusters such that the set of classes provide essentially the same information about the image archive as the entire image-set collection. Image clustering enables the implementation of efficient retrieval algorithms.

● Image Resources: Nature and Human Image

The following images are used for applying clustering techniques in this tutorial


Lotus picture


Jennifer Lawrence

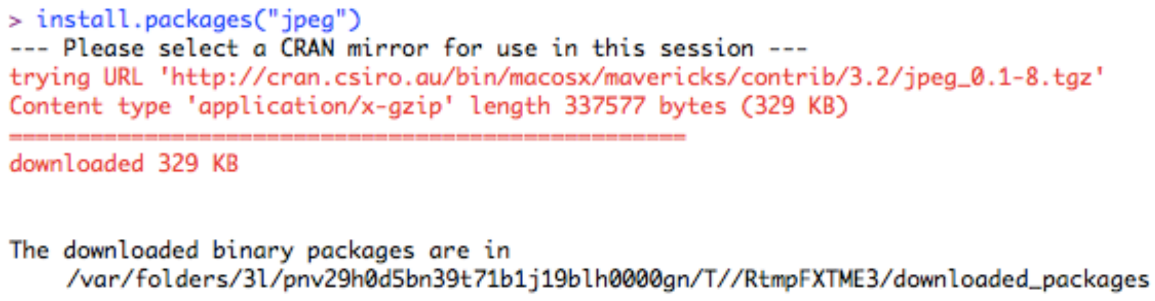
By using clustering technique, we would like to solve the following issues:
1. Image features – how to represent the image.
2. **Organization of feature data – how to organize the data.**
3. Classifier – how to cluster an image to a certain group.

Image clustering is the process of partitioning a **digital image into multiple segments** (sets of pixels, also known as super pixels). The goal of segmentation is **to simplify and/or change the representation of an image** into something that is more meaningful and easier to analyse.

We will apply this method to an image, wherein we group the pixels into *k* different clusters.

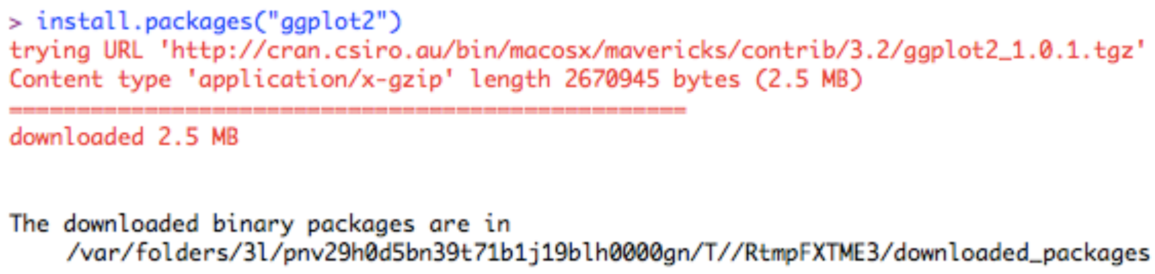
● Required Packages for Image Clustering: jpeg, ggplot2

It is crucial to install the following packages for handling input and output in image clustering.
1. jpeg – read and write Graphics devices for BMP, JPEG, PNG and TIFF format bitmap files.
2. ggplot2 – An Implementation of the Grammar of Graphics



```
> install.packages("jpeg")
--- Please select a CRAN mirror for use in this session ---
trying URL 'http://cran.csiro.au/bin/macosx/mavericks/contrib/3.2/jpeg_0.1-8.tgz'
Content type 'application/x-gzip' length 337577 bytes (329 KB)
==================================================
downloaded 329 KB


The downloaded binary packages are in
    /var/folders/3l/pnv29h0d5bn39t71b1j19blh0000gn/T//RtmpFXTME3/downloaded_packages
```

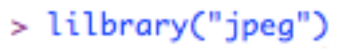


```
> install.packages("ggplot2")
trying URL 'http://cran.csiro.au/bin/macosx/mavericks/contrib/3.2/ggplot2_1.0.1.tgz'
Content type 'application/x-gzip' length 2670945 bytes (2.5 MB)
==================================================
downloaded 2.5 MB


The downloaded binary packages are in
    /var/folders/3l/pnv29h0d5bn39t71b1j19blh0000gn/T//RtmpFXTME3/downloaded_packages
```

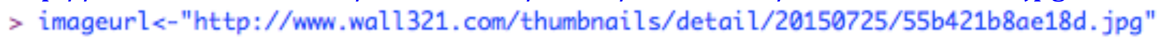
● Downloading and Reading Image File

Now, we installed required packages for image reading and writing. By using jpeg library, it is required to download the image into your computer, and command read the image file as jpeg.
We should load the library that installed before



```
> lilbrary("jpeg")
```

Define the url of the image. The first image we are using is flower (lotus)
http://www.wall321.com/thumbnails/detail/20150725/55b421b8ae18d.jpg



```
> imageurl<-"http://www.wall321.com/thumbnails/detail/20150725/55b421b8ae18d.jpg"
```

Download the image file to your workspace using the following command:

```
> dFile <- download.file(imageurl, "lotus.jpg")
trying URL 'http://www.wall321.com/thumbnails/detail/20150725/55b421b8ae18d.jpg'
Content type 'image/jpeg' length 126807 bytes (123 KB)
==================================================
downloaded 123 KB
```

Read the downloaded image. The downloaded image is in JPEG or JPG format so we use the readJPEG function. The **readJPEG** function allows users to read an image from a JPEG/content into a raster array.

```
> img <- readJPEG("lotus.jpg")
```

In **readJPEG**, there is argument 'native', which determines the image representation - if FALSE (the default) then the result is an array, if TRUE then the result is a native raster representation.

If native is FALSE then an array of the dimensions height x width x channels. If there is only one channel the result is a matrix. The values are reals between 0 and 1. If native is TRUE then an object of the class nativeRaster is returned instead. The latter cannot be easily computed on but is the most efficient way to draw using raster Image. Most common files decompress into RGB (3 channels) or Grayscale (1 channel). Note that Grayscale images cannot be directly used in rasterImage unless native is set to TRUE because rasterImage requires RGB or RGBA format (nativeRaster is always 8-bit RGBA). JPEG doesn’t support alpha channel, you may want to use PNG instead in such situations.

● Preprocessing the Image Data

From the result of reading image file, it is necessary to extract and organise the useful information for the image clustering. First, it is necessary to obtain the dimension for the image.

```
> imgDm <- dim(img)
```

After the dimension is extracted, it is necessary to assign x and y location (from the extracted dimension, and each RGB channel (from the related pixel in image file) into data frame.

```
> imgRGB <- data.frame(
+   x = rep(1:imgDm[2], each = imgDm[1]),
+   y = rep(imgDm[1]:1, imgDm[2]),
+   R = as.vector(img[,,1]),
+   G = as.vector(img[,,2]),
+   B = as.vector(img[,,3])
+   )
```

The image is represented by large array of pixels with dimension *rows* by *columns* by *channels* — red, green, and blue or RGB.

You can see the developed imgRGB dataframe as follows:

```
> imgRGB
         x   y          R         G           B
1        1 337 0.34117647 0.4901961 0.305882353
2        1 336 0.35294118 0.5019608 0.301960784
3        1 335 0.36078431 0.5098039 0.286274510
4        1 334 0.38039216 0.5137255 0.274509804
5        1 333 0.39215686 0.5176471 0.262745098
6        1 332 0.40392157 0.5254902 0.262745098
7        1 331 0.40000000 0.5254902 0.239215686
8        1 330 0.40784314 0.5333333 0.207843137
9        1 329 0.41568627 0.5294118 0.168627451
10       1 328 0.41960784 0.5294118 0.129411765
```

- **Set a theme for drawing the image**

From the previous steps, we completed downloading and reading image, and assigning the dimension, RGB channel for each pixel. Now it is time to plot the image in R console using extracted imgRGB.

Call ggplot2 library.

```
> library(ggplot2)
```

The plot theme (frame of your image) should be defined before you plot the image using ggplot function. The following commands use the function **theme. The detailed information for each argument can be found as follows:**

```
> plotTheme <- function() {
+   theme(
+     panel.background = element_rect(
+       size = 3,
+       colour = "black",
+       fill = "white"),
+     axis.ticks = element_line(
+       size = 2),
+     panel.grid.major = element_line(
+       colour = "gray80",
+       linetype = "dotted"),
+     panel.grid.minor = element_line(
+       colour = "gray90",
+       linetype = "dashed"),
+     axis.title.x = element_text(
+       size = rel(1.2),
+       face = "bold"),
+     axis.title.y = element_text(
+       size = rel(1.2),
+       face = "bold"),
+     plot.title = element_text(
+       size = 20,
+       face = "bold",
+       vjust = 1.5)
+   )
+ }
```

- panel.background: background of plotting area, drawn underneath plot (element_rect; inherits from rect)
- axis.ticks: tick marks along axes (element_line; inherits from line)
- panel.grid,major: major grid lines (element_line; inherits from panel.grid)
- panel.grid.minor: minor grid lines (element_line; inherits from panel.grid)
- axis.title.x: x axis label (element_text; inherits from axis.title)
- axis.title.y: y axis label (element_text; inherits from axis.title)
- plot.title: plot title (text appearance) (element_text; inherits from title)

Try to change the value of each argument and see the differences.

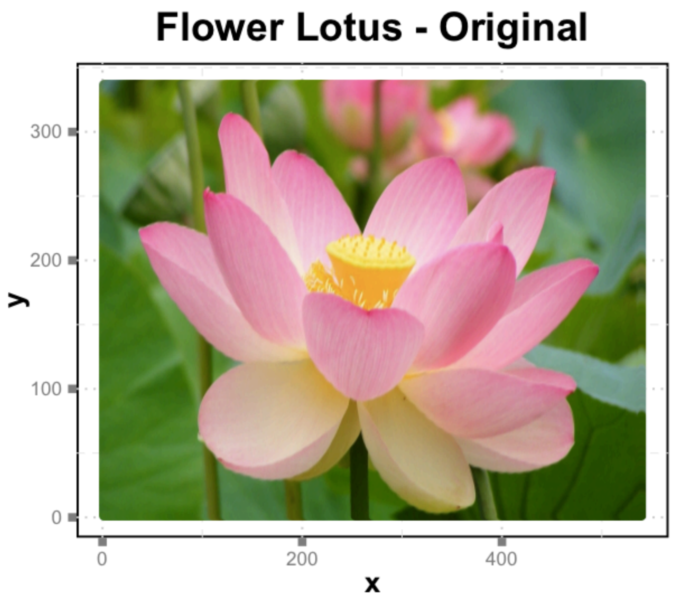
- **Plotting the image using the theme developed in the previous step.**

Before clustering the image, let's try to plot the original image using the following commands. We use the function ggplot(), which initializes a ggplot object. It can be used to declare the input data frame for a graphic and to specify the set of plot aesthetics intended to be common throughout all subsequent layers unless specifically overridden.

geom_point is the function, which is used to create scatterplot. In this case, the data point for our domain is included in the imgRGB variable.
labs, xlab, ylab, and plotTheme function will be used based on how you define the theme in the above command.

```
> ggplot(data = imgRGB, aes(x = x, y = y)) +
+   geom_point(colour = rgb(imgRGB[c("R", "G", "B")])) +
+   labs(title = "Flower Lotus - Original") +
+   xlab("x") +
+   ylab("y") +
+   plotTheme()
```

NOTE: Plotting image may take some time (less than 10 seconds) depends on your computer.



● Image Clustering using K-means Clustering

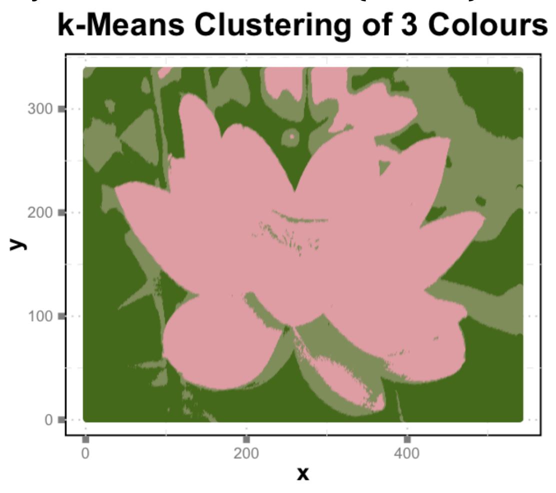
We learned how to download image, convert the image into data point, and plot them. Now, it is time to apply k-means clustering to distinguish the groups (i.e. floral leaf, stamen, or stem/stalk) in the following image.

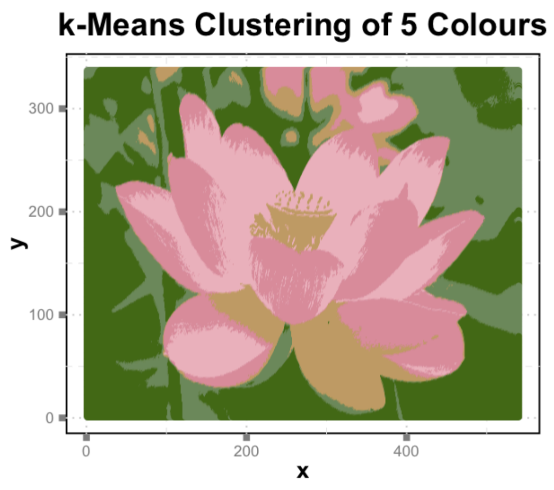
The **k-means algorithm** is an algorithm to cluster n objects based on attributes into k partitions, where k < n. The process of k-means clustering can be found from the last tutorial (tutorial 7).

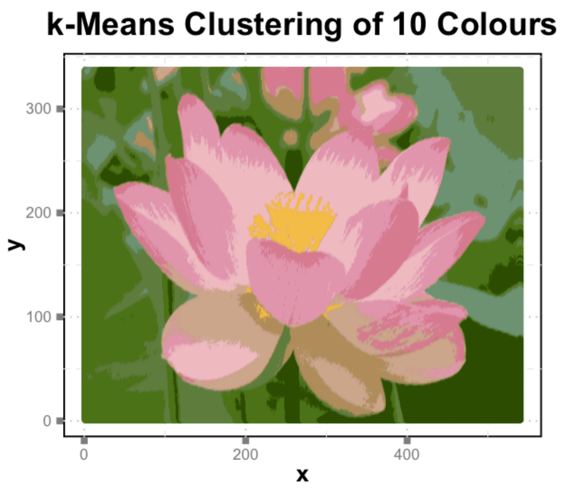
So, at the first time, we set up the number of K cluster as 3. The k-means clustering uses the dataset imgRGB, and the centre should be shown with the rgb colour of the defined centre
Then you can plot the data points using ggplot function.

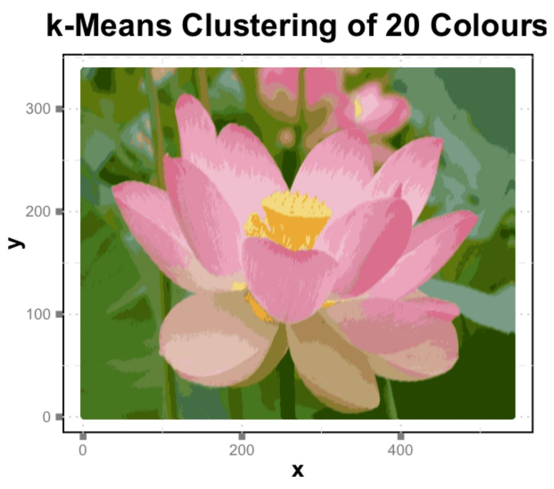
```
> kClusters <- 3
> kMeans <- kmeans(imgRGB[, c("R", "G", "B")], centers = kClusters)
> kColours <- rgb(kMeans$centers[kMeans$cluster,])
>
> ggplot(data = imgRGB, aes(x = x, y = y)) +
+   geom_point(colour = kColours) +
+   labs(title = paste("k-Means Clustering of", kClusters, "Colours")) +
+   xlab("x") +
+   ylab("y") +
+   plotTheme()
```

Try different number of k (5,10,20) and check the result









Tutorial 8 Question

The data we will use for tutorial 8 is Jennifer Lawrence Image.

(http://www.wall321.com/thumbnails/detail/20150816/55d03e1abc946.jpg)

The required packages for this tutorial 8 are as follows:

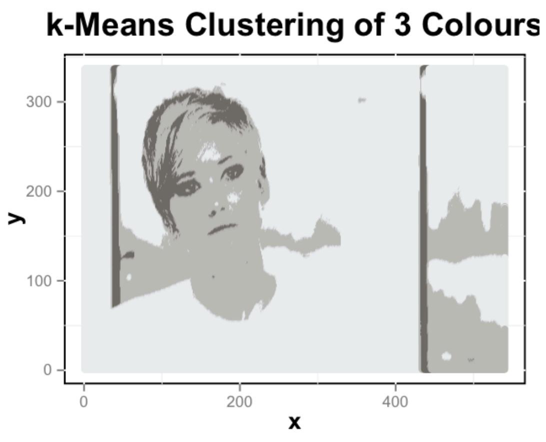
- **jpeg**
- **ggplot2**

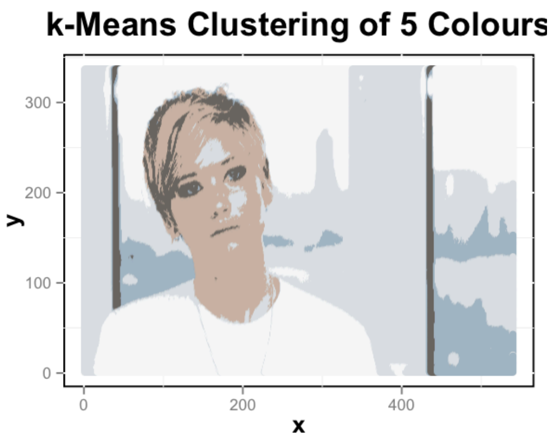
1. **Download the image and read the image file (using readJPEG)**

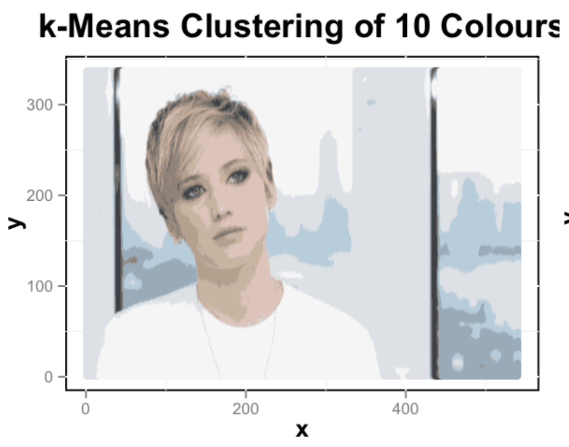
2. **Obtain the dimension, and assign x, y, and RGB channels to data frame**

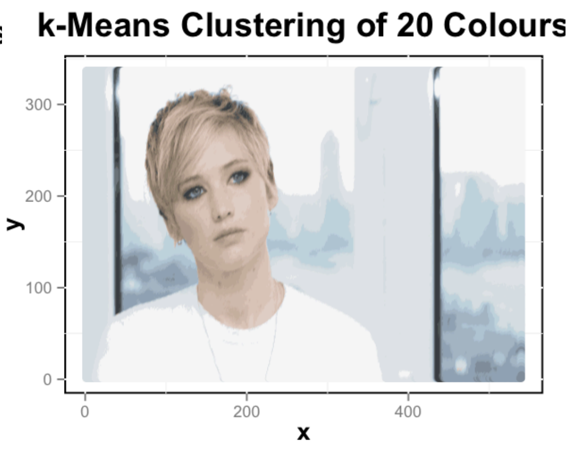
3. **Set up the theme for the image plotting with the following conditions**
   - panel.background : element rect size=1, colour=”black” and fill=”white”
   - axis.title.x: element text size=rel(1.2), face=”bold”
   - axis.title.y: element text size=rel(1.2), face=”bold”
   - plot.title: element text size=20, face=”bold”, vjust=1.5

4. **Set up K as 3, 5, 10, 20 and compare the outcome. Discuss the minimum plot that can distinguish background, head, and body from the image.**






k-Means Clustering of 10 Colours
y
300
200
100
0
0
200
400
x


k-Means Clustering of 20 Colours
y
300
200
100
0
0
200
400
x