MATH336 Group Project

Group J

Jake Amo, Michael Jennings, James Robinson

First Meeting Summary

Attendees

- Jake Amo
- James Robinson
- Michael Jennings

In our first meeting, we tried to gain a good understanding of what our investigation was going to involve and how we could break it down into smaller tasks such that we could all have our influence on the final project. After a long productive talk, we were able to work out what we already knew and what we had to learn as a group so that we could start to construct some structure to our work. So our first plan was for all of us to go away and do some more profound research into the topic, specifically, we all wanted to understand the theory behind the algorithm as we all thought this was fundamental in completing all of the necessary tasks. After we had done this, we regrouped and started to delegate the work. As a provisional, we started with these tasks:

Jake Amo - K-means applied to image compression
James Robinson - The Clustering Problem
Michael Jennings - writing the pseudocode and describing the procedures, running the examples

but after we started, we were all keen to get involved in each task, and we all familiarised with each others' task.

0.1 Introduction

Cluster analysis, also known as "clustering", aims to group similar observations. Such observations (our data points) are grouped into "clusters" in a way that the objects within the same cluster are more similar to objects in others.

We seek to partition our data frame in K disjoint subsets, our clusters, where every element of the data frame belongs uniquely to one cluster, and the union of all clusters is hence the data frame itself [1].

According to the various clustering techniques the cluster membership of each observation can be either probabilistic, *i.e.* we define a probability for each observation belonging to each of the clusters, called *soft partition clustering*, or it can be strict, where each observation is assigned to a specific cluster only, known as *hard partitioning clustering*.

Applications of cluster analysis can be found through many fields that employ the use of data mining or statistical data analysis, such as in pattern recognition, machine learning, image analysis, and data compression. For the sake of this paper, we focus on applying clustering to data compression, using k-Means, which is a distance-based hard partitioning clustering method.

We will start by describing the clustering problem and the k-Means procedure itself. After considering k-Means with its features and drawbacks, we provide a description of how the problem can be applied to image compression.

0.2 The Clustering Problem

As a start, we point out that the clustering problem is essentially different from other classification problems. Those methods, such as linear modelling, neural networks or support vector machines, use a set of predictors to infer the value of a known variable, which we call the outcome. This means that we have already some information on the outcome itself, our variable of interest, and we try to learn the connection between such variable and the remaining predictors. As for when the outcome variable is unknown, those methods can be used to predict our outcome after we have trained our model on a data frame for which the outcome variable is known. For this reason, these methods solve what is called a *Supervised learning* problem.

Clustering is essentially different as it does not require any pre-known information about our data, nor it is trying to predict an outcome variable. That means that clustering identifies similarities between data points and assigns them into classes it creates based off these similarities. For this reason, in machine learning, we say that clustering is an *unsupervised learning* problem. Hence the outcome of interest is the cluster belonging of each observation itself alongside a description of these clusters.

To begin formulating an algorithm to cluster our data, we must first define what we want our cluster to be. This is a difficult concept itself, however, as the notion of a cluster cannot be formally defined (or, more specifically, we can define a cluster in several ways) [2]. Consider the figures below:

Figure 1: 3 separate data plots with clear clusters that can easily be defined [1].

In figure 1, we can easily spot clusters in the data, either because the data is already clustered into groups (such as in the latter two examples), or because most of the data is already centred around two clusters (such as in the first plot). In these cases, it is not complicated to figure out what a cluster could be.

However, in figure 2, one could notice that by drawing any such arbitrary line withing the cloud of observations, we could come up with any reasonable clustering. This is an example of the ambiguity that can arise in clustering problems, as one could even argue that in the figure we have only one cluster itself and hence the data doesn't even

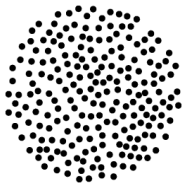

Figure 2: A data plot with seemingly no clear clusters that can be easily defined [1].

have a need to be clustered. The lesson we learn from such cases is that according to the situation, we can define clusters based on what our data is, what outcome we are looking for, or for what purpose we are using clustering.

For the sake of this report, we will define a cluster by a cloud of observations around a *centroid*, $c_i \in \mathbb{R}^p$. This can either be one of our data points or some 'arbitrary' point. This essentially involves defining a distance, in order to be able to classify observations and give a cluster belonging based on which centroid is the closest. We find in the literature a plethora of ways to define a distance between two points. The most commons are the Euclidean, Manhattan, or Minkowski distances [3].

As mentioned before, there exist other definitions of a clusters, such as in density-based clustering, where we look at clusters of high point density regions separated by low point density regions, or the distribution-based clustering, where we look at the likelihood of a point belonging to a certain cluster [1].

In the next section of this paper, we consider only centroid-based clustering, more specifically the k-means procedure. After an introduction to the general algorithm, we then provide a description of how this can be utilised to perform image compression.

0.3 k-Means Clustering and K-means for image compression

The process begins by selecting K distinct data points. We choose these to be our initial centroids labelled c_i, each defining a different cluster and assign each observation to each cluster based off the Euclidian distance, $d(x, y)$. Let $x, y \in \mathbb{R}^p$ we calculate this as follows:

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_p - y_p)^2}$$

We assign each data point to the cluster of its nearest centroid, that is, we aim to minimize the Euclidian distance between centroids $c_1, ..., c_K$ and data points $x_1, \ldots, x_n$.

In the second iteration of the algorithm, we now find the mean of the data points in each of the K clusters and treat this mean as if it were a new set of centroids, repeating the process of allocating all the data points to their new clusters. The procedure is detailed in 0.3.1.

Thus, for K-means clustering, the most important decision is the value of K, as it is going to determine how many clusters you form. In some cases, such as Figure 1 the value of K may be clear; in other cases, it's not, and therefore you may need to test different values for K.

So what constitutes good clustering? Effective clustering can be described as minimising the value of K while still maintaining high quality. As K increases, so does the quality of clustering. In particular for image compression, a higher value of K leads to an image with higher colour depth. However, there may be trade-offs, such as greater file size.

How can we measure the quality of a cluster? One technique is collecting multiple results of the procedure using different K values and comparing them. In this case, we would be interested in minimising the total distance of each point to the centroid of the cluster it is in. We can do this by summing the variation within each cluster.

As it can be seen from the procedure, the first step is aleatory as it involves a random centroid initialisation. It is

important to note that different initialisation may result in different K clusters. In these cases, it may be better to manually choose specific centroids with the technique above which will produce more accurate results.

0.3.1 K-means Clustering pseudo-code

Below, we give a formally defined algorithm for k-means clustering, using the ideas we discussed in the above section:

Input $X \in \mathbb{R}^{n \times p}$ a matrix with n observations in p variables.
Initialization Select K rows at random (or by selection) and initialize them as the K unique centroids, $c_1, \ldots, c_K$. These centroids will now form clusters (groups) $G_1, \ldots, G_K$ which, for the moment, only contain the row which created them.

Step 1 For each centroid $c_k \in \mathbb{R}^p,\ k = 1, ..., K$ and for each observation $x_i \in \mathbb{R}^p,\ i = 1, ..., n$, find the distances $d(c_k, x_i)$ between them. Then assign each observation to the nearest cluster $G_1, \ldots, G_K$ based on which has the smallest distance $d(c_k, x_i)$.
Step 2 For each cluster $G_1, \ldots, G_K$, calculate and update the centroids $c_1, \ldots, c_k$ by averaging value of each component of the vectors within each specific group. *E.G.*:

$$c_k = \frac{1}{n_{Gk}}(\sum_i x_{i1},\ \sum_i x_{i2},\ \ldots,\ \sum_i x_{ip})$$

where n_{Gk} is the number of elements withing cluster G_k.
Step 3 Re-assign every point into the cluster which centroid it is closest to. If any group assignment has changed then return to **Step 2**, otherwise set each row equal to the centroid of the cluster it is in. Return $G_1, \ldots, G_k$.

0.3.2 K-means applied to image compression pseudo-code

In the following section, we will describe how the k-means procedure can be applied to image compression starting from the input, a 24-bit colour image.

Input the image $I \in \mathbb{R}^{h \times w \times 3}$ to be a 3-dimensional matrix
Initialization

- h: is the height of the image;
- w: is the width of the image;
- and finally we have 3 planes each corresponding to the Red, the Green and Blue values for each pixel.

Step 1 Flatten the 3-dimensional matrix into a 2-dimensional data frame $X \in \mathbb{R}^{(h \cdot w) \times 3}$ where each row now represents one pixel and the columns represent the Red, Green and Blue colour value.
Step 2 Run the K-means procedure, as described in Section 0.3.1, on the data frame X with K equal to the number of colours you now want to represent the image.
Step 3 For your k clusters $G_1, \ldots, G_k$ achieved by running the k-means algorithm, you can now reconstruct the image with the new colour values for each pixel.

Note How step 1 and step 3 are performed is a language-specific feature. Hence we do not provide a full description in the pseudo-code.

In R, this can be easily achieved by recording the position of each pixel in the data frame and by plotting each pixel colour value over the relative x and y positions. We only input the columns representing the Red, Green and Blue values into the K-means algorithm as the position of the colour is irrelevant to achieving the best K-means values in image compression.

We now apply the k-means algorithm with $k = 16$ to a selected colourful image and compare the results between the original and the compressed images.

Figure 3: Original Image

Figure 4: Image after k-means algorithm applied

At first glance, it is clear that both images are quite similar; however, upon closer inspection, differences become very apparent. The compressed image's colours appear to be less saturated than the original's giving the compressed image a dimmer feel, this is due to the final centroids having settled on darker tones, and is most apparent when comparing the whites of the lights on the lampposts of both images.

After the algorithm, the image also loses its softness, the colours rather than transitioning between tones jump from one to the other abruptly. This is to be expected given the algorithm has been limited by its value of k, so colours such as orange jump straight to brown as there are not enough distinct clusters (different colours) to allow for a transition between the two.

Conclusion

From the result, we can see that the compressed image after being processed by the k-means algorithm with a k value of 16 gives a reasonably accurate representation of the original. However, there are some clear differences between the two images, this is primarily noticeable around the centre and centre top of the image, where the compressed image fails to identify the light greens and tones of pink the original image has. The quality of the image would clearly increase with a higher value of k, perhaps with $k = 17$ or $k = 18$ we would be able to see these discrepancies. However, the trade-off would be a larger file size for a minor difference.

The impact of the increase of k diminishes over increments, namely, the difference in the image would be more significant from $k = 3$ to $k = 4$ then from $k = 19$ to $k = 20$ which means you can quantify an optimal value of k however this usually cannot be done manually.

One of the main limitations of k-means clusters is the dependence on the initial centroids, in cases like ours where k is small this is not that big of a problem, but in problems where you have bigger values of k, it becomes difficult to run the algorithm enough times so as to get good initial centroids for your clusters.

In conclusion, it can be seen that the k-means algorithm along with our given value of $k = 16$ gives a suitable compression of this image.

Bibliography

[1] Brian Everitt. *Cluster Analysis*. Wiley, 5th edition, 2011.

[2] Vladimir Estivill-Castro. Why so many clustering algorithms - a positioning paper, 2002.

[3] Santosh Kumar Uppada. Centroid based clustering algorithms - a clairon study, 2014.

Appendix

R code for K-means image compression

```
# Read the image
img <- readJPEG(image.path)

# Obtain Image  dimension
imgDm <- dim(img)

# Assign RGB channels to data frame
# we are flattening the tri dimentional matrix onto a two dimentional matrix
# by having each pixel on a row and recording its relative values of R G B
imgRGB <- data.frame(
  x = rep(1:imgDm[2], each = imgDm[1]),
  y = rep(imgDm[1]:1, imgDm[2]),
  R = as.vector(img[,,1]),      # we take the red component and we flatten it into one vector
  G = as.vector(img[,,2]),      # we take the green component and we flatten it into one vector
  B = as.vector(img[,,3])       # we take the blue component and we flatten it into one vector
)

# so at the end, our flattened matrix will have a pixel each row and for each pixel, a red, a
# green and a blue value

# Obtain a plot of the original image
plot1<-ggplot(data = imgRGB, aes(x = x, y = y)) + geom_point(colour = rgb(imgRGB[c("R", "G", "B")]))
+ labs(title = "Original Image")

# Number of clusters
k <-16

# Compress the image using k-means clustering
kMeans <- kmeans(imgRGB[, c("R", "G", "B")], centers = k)
num.of.colours <- rgb(kMeans$centers[kMeans$cluster,])

# as an example, we compute the first value of the first centroid (red value)
```

```
mean(imgRGB[kMeans$cluster == 1, "R"]) # for read
mean(imgRGB[kMeans$cluster == 1, "G"]) # for green
mean(imgRGB[kMeans$cluster == 1, "B"]) # for blue

# Obtain a plot of the compressed image
plot2<-ggplot(data = imgRGB, aes(x = x, y = y)) + geom_point(colour = num.of.colours)
+ labs(title = paste("k-Means Clustering of", k, "Colours"))

# Plot the original and compressed image side by side
grid.arrange(plot1, plot2, nrow=2)
```