k-means Clustering in Machine Learning

Sebastian Knight

November 12, 2018

Contents

1 Introduction

In the following report we will be looking at k-means clustering as an example of machine learning.

We begin by discussing what machine learning is and the differences between supervised and unsupervised learning. We then proceed to introduce k-means clustering as an example of an unsupervised learning algorithm. An example is then given of the use of k-means clustering in reducing the size of an image by reducing it's colour count, before a conclusion is given on the uses and usefulness of k-means clustering as an unsupervised learning algorithm.

2 Machine Learning

In a general sense, "learning" is the act of converting experience into expertise. Taking known data and extrapolating to apply what we learn from it to more general circumstances. Machine learning is similar. It takes a set of data and uses it to form a hypothesis which it can then be applied to a more general data. generating a correct answer on new data points with a high probability of being approximately correct.

One of the biggest issues with machine learning is that it can never be 100% correct as it is generates the hypothesis based on a limited data set and, as such, there is no guarantee that further data points outside that training set coincide with those points at all. Thus machine learning can only ever be *Probably Approximately Correct* (PAC).

There are several types of machine learning. Two major classifications of learning are *supervised* and *unsupervised* learning.

Supervised: In Supervised learning the training date comes in the form of paired examples of input and their corresponding correct output.
For example; one machine learning problem is to take candidates applying for credit cards and accept or reject their application based on a number of factors, such as income, savings, ect. In this example the program would be given a training set of candidates who have applied in the past and whether or not they were accepted. Thus each point in the training set will have an attached label. The "answers" are known.
Unsupervised: In the case of unsupervised learning the training set does not include any information about the correct labels. Going back to our above example, the training set would be formed of the candidates and their input data (income, savings, ect.) but not their corresponding labels. The program would then have to construct a rule by which it could label new applicants.
One way that this can be done is by clustering. Attempting to group the data points into different clusters depending on their input data distribution.
K-means clustering is one such algorithm that does just this. k-means clustering is an algorithm that can be used with an unsupervised data set. The simple, or standard, k-means algorithm works by finding a set of k means such that each data point is in a cluster surrounding a closest mean.

The k-means algorithm works by reiterating several steps[1] (see figure1).

1 k initial "means" (in this case $k = 3$) are randomly generated within the same space as the data (shown in color).

2 k clusters are created by associating every observation with the nearest mean.

3 The centroid, or central point, of each of the k clusters becomes the new mean.

4 Steps 2 and 3 are repeated until convergence is been reached.

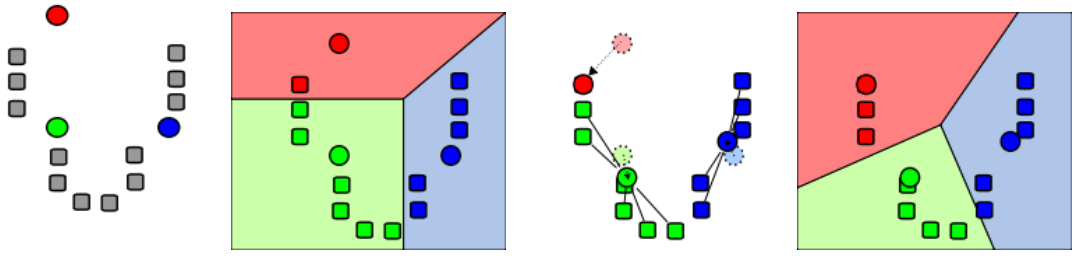

Figure 1: An example of the 4 steps of the k-means algorithm converging withing a data set.

One of the major problems with the k-means algorithm is that it finds a local minimum rather than a global one. When applied multiple times to the same data set using large k values the outcome depends on the initial positions of the K means[1]. So there is a possibility of it grouping a data set incorrectly when a more obvious clustering set exists.

3 Image Compression

One possible application of the k-means clustering algorithm is to reduce the size of image files so that they are faster and easier to transfer. Image files are composed of thousands of pixels and each pixels is represented as three 8-bit integers between 0 and 255 indicating it's RGB value. This can mean that a single image can contain thousands of colours, with each pixel being assigned a unique value, which can make them time consuming to transfer. One way to reduce this is to restrict the number of colours in the image. Then, instead of each pixel being given three values between 0 and 255, they can be given a single value between 0 and k that represents their colour in it's entirety.

We will be using a k value of 16 in this example. Thus the algorithm will group the pixels into clusters according to 16 to-be-determined means dependent on which mean is "closest" to each point.

1. First, we take an image(figure 3 in Appendix A) and convert it to a data frame consisting of each pixels RGB value.
2. We then apply the k-means algorithm to this data frame.
3. Finally, we plot both images side by side to give a comparison.(figure 2)

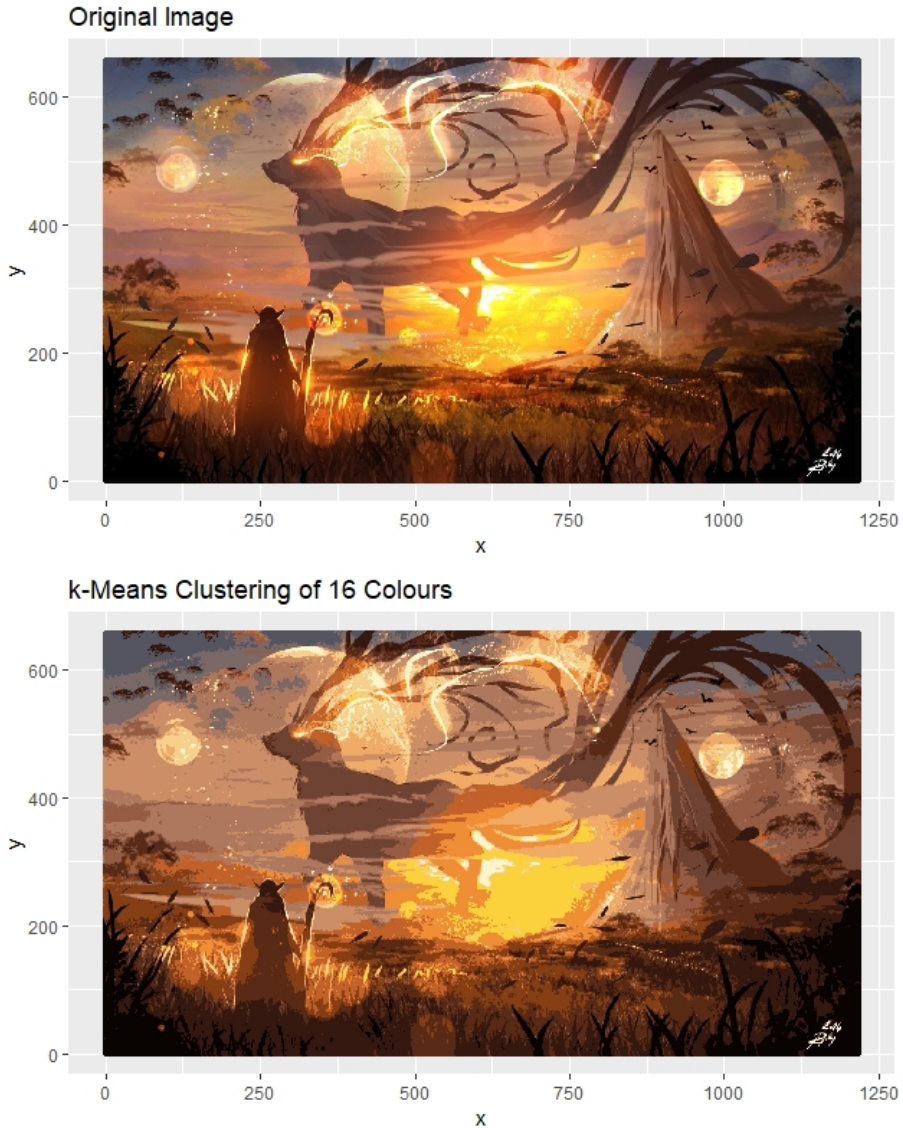

Figure 2: A comparison of the original image (top) with the image after the application of the k-means algorithm, using k=16

As can be seen, the image after the application of the algorithm has much less fidelity than the original. Many of the intricacies of the design have been blocked together and origional smooth transitions due to light gradient are now blocky and sudden. Having said that, it did manage to capture a surprising amount of detail, such as the wisps of light from the eyes as well as the birds over the mountain.

A sort study into the effects of varying the value of k gave visible results (see figure 4 in Appendix A) increasing the value of k further increases the fidelity of the resulting image, capturing more of the finer details and producing a smother gradient between colours and shading. Whereas lowering the value of k produces a washed out image where only the broadest sense of the original is preserved. The most notable difference between the varying values of k is the accuracy to which it captures the details when there is only slight colour differences at the borders between shading cells. With lower values of k those colours are clustered together, resulting in the almost silhouetting of some of the figures in the image. Whereas with higher values almost all of the inner details of the figures is preserved.

4 Conclusion

In conclusion, k-means clustering is a surprising good method of compressing an image as, even with the larger values of k, the number of colours needed is significantly smaller than that of the original. Furthermore, especially with the higher values of k, the actuary to the original was high. Many of the same shapes and gradients were still present, and thus the same "feel".

However, k-means clustering does have it's downsides. It is inconsistency, with the outcome depending on the initial values of k, allowing it to incorrectly cluster some data sets on occasion. No further testing was done during this project, but it is probable that is the algorithms were to be run again, different levels of fidelity would result, perhaps even changing the conclusions presented here.

Overall, k-means clustering is a simple machine learning algorithm that can, with caution, be applied to a variety of unsupervised learning problems with reasonable confidence that it will be probably approximately correct. Though is more security is required, more accurate clustering algorithms exist that can be used in it's stead.

5 Appendix A: R Code

```
install.packages("jpeg")
install.packages("gridExtr")

library(jpeg)
library(ggplot2)
library(grid)
require(gridExtra)

image.path <- paste(getwd(), "/R/Image.jp", sep = "")# This gives path to image
img <- readJPEG(image.path)# Read the image

# Obtain Image dimension
imgDm <- dim(img)

# Assign RGB channels to 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])
)

# 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 = "OriginalImage")
plot1b <- ggplot(data = imgRGB1, aes(x = x, y = y))+geom_point(colour = rgb(imgRGB1[c("R", "G", "B")]))+
labs(title = "OriginalImage")

# Compress the image using k-means clustering
k <- c(3, 16, 32)#Numberofclusters
kMeans1 <- kmeans(imgRGB[, c("R", "G", "B")], centers = k[1])
kMeans2 <- kmeans(imgRGB[, c("R", "G", "B")], centers = k[2])
kMeans3 <- kmeans(imgRGB[, c("R", "G", "B")], centers = k[3])
num.of.colours1 <- rgb(kMeans1$centers[kMeans1$cluster, ])
num.of.colours2 <- rgb(kMeans2$centers[kMeans2$cluster, ])
num.of.colours3 <- rgb(kMeans3$centers[kMeans3$cluster, ])

# Obtain a plot of the compressed image
plot2 <-ggplot(data = imgRGB, aes(x = x, y = y)) + geom_point(colour = num.of.colours1) +
labs(title = paste("k - MeansClusteringof", k[1], "Colours"))
plot3 <-ggplot(data = imgRGB, aes(x = x, y = y)) + geom_point(colour = num.of.colours2) +
labs(title = paste("k - MeansClusteringof", k[2], "Colours"))
plot4 <-ggplot(data = imgRGB, aes(x = x, y = y)) + geom_point(colour = num.of.colours3) +
```

$labs(title = paste(\text{“}k - MeansClusteringof\text{”}, k[3], \text{“}Colours\text{”}))$

Plot the original and compressed image side by side
$grid.arrange(plot1, plot2, nrow = 2)$
$grid.arrange(plot1, plot3, nrow = 2)$
$grid.arrange(plot1, plot4, nrow = 2)$
$grid.arrange(plot1, plot2, plot3, plot4, nrow = 2)$

6 Appendix B: Images

Figure 3: The original image before application of the k-means algorithm

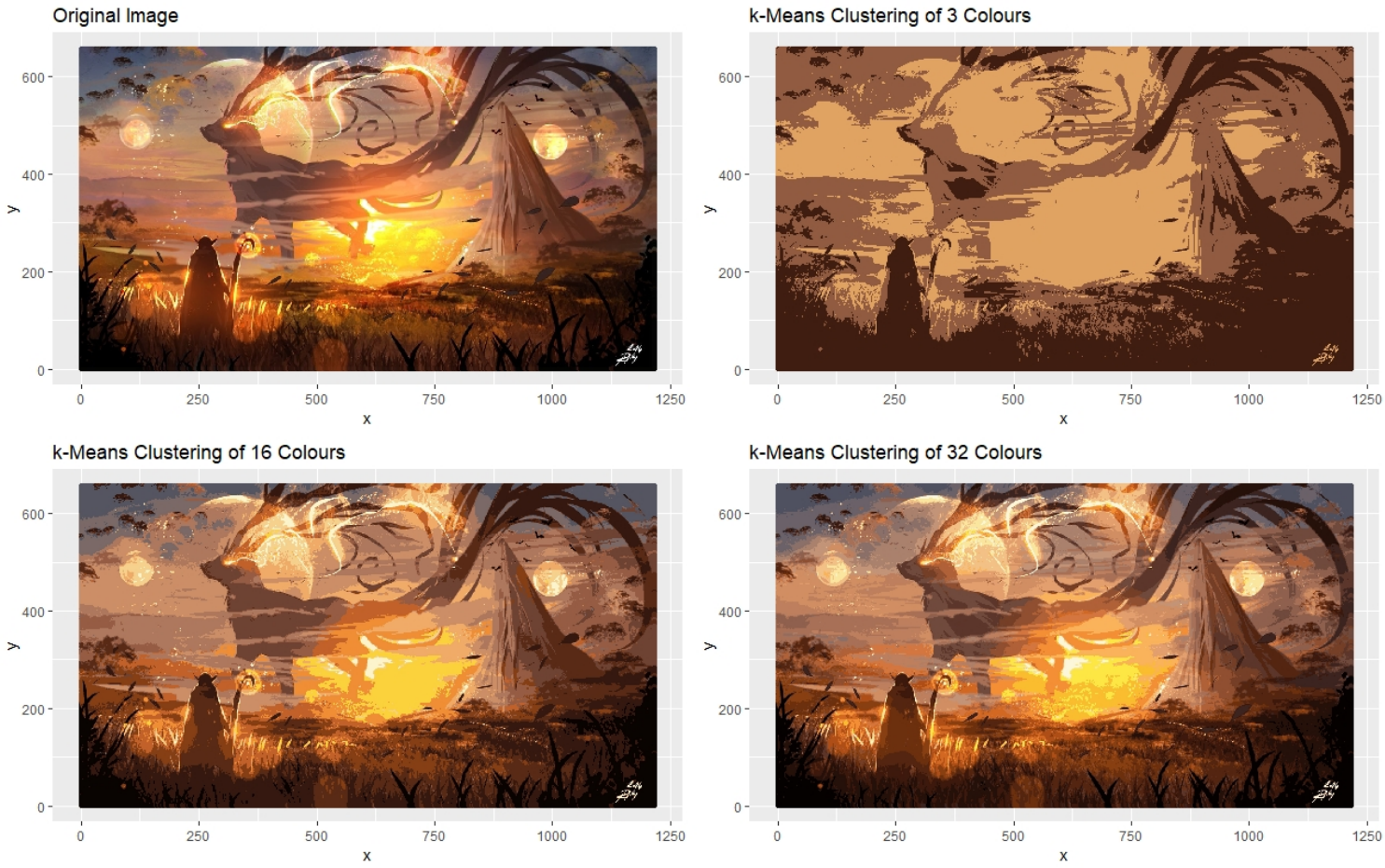

Figure 4: A side by side comparison of the differences between k values of 3,16 and 32 in k-means clustering

7 References

1. MacKay, David (2003). "Chapter 20. An Example Inference Task: Clustering." Information Theory, Inference and Learning Algorithms. Cambridge University Press. pp. 284&ndash, 292. ISBN 0-521-64298-1. MR 2012999.
2. The immage used in this project is a pice called "Something Beautiful" by ryky of DiviantArt.