INDIAN INSTITUTE OF TECHNOLOGY KANPUR

MTH552A

AI & STATISTICAL TECHNIQUES IN DATA MINING

Colour Identification using K-Means Clustering

Authors
Abhimanyu Sethia
Aayush

Supervisor
Prof. Amit Mitra

April, 2022

1 Introduction

For humans, the perception of colour is integral to sensing the environment, conveying information and recognizing objects. In our endeavour to make computer systems intelligent enough to extract information from images like humans do, we need to build the robust ability to identify colours in an image.

This project attempts to apply K-Means Clustering for identifying colours in images. A colour identifier has wide-ranging applications from real-time colour sensors used in Cyber-physical systems (CPS) to developing colour-based image filters for applications like Google Photos. Since our endeavour was to understand the math behind how K-Means clustering works in real life, we have abstained from using any external ML libraries and have coded our K-Means algorithm from scratch.

2 K-Means Clustering

K-means clustering is based on the idea that each point in a cluster should be near to the center of that cluster. The algorithm tries to learn an optimal division of space to group the data-points into K different classes. K-means clustering is an unsupervised learning algorithm.

2.1 Formal Statement of the Clustering Problem

In any clustering problem, given n data points $\{x_{(1)}, \cdots, x_{(n)}\}$, we need to find k clusters, characterised by their centroids $(\mu_1, \cdots, \mu_k)$ and assign each data point i to exactly one cluster $\pi(i) \in \{0, 1, \cdots, k-1\}$. Here, $x_i \in \mathbb{R}^m$ where m is the dimensionality of the Euclidean space.

It may be noted that since this is an unsupervised learning problem, no labels $y^{(i)}$ are given.

2.2 K-Means Algorithm

Let $(\mu_1^{(i)}, \cdots, \mu_k^{(i)})$ be the centroids of the k clusters at the ith iteration. Here is the pseudo-code for the k-Means algorithm

- Initialize k cluster centroids $(\mu_1^{(0)}, \cdots, \mu_k^{(0)})$ randomly
- For j from 1 to max number of iterations
  1. Cluster Assignment for all points: For i from 0 to $n-1$,
  $$\pi(i) = argmin_{r \in \{1, \cdots, k\}} ||x_i - \mu_r^{(j)}||^2$$
  2. Centroid Updation for all clusters: For r from 0 to $k-1$,
  $$\mu_r^{(j+1)} = \frac{1}{|\{i : \pi(i) = r\}|} \sum_{i:\pi(i)=r} x_i$$

2.3 Analysis

In the algorithm, we basically carry out two steps on loop- (i) assigning each training example x_i to the cluster whose centroid is closest (by Euclidean distance) to the x_i and (ii) Updating each cluster centroid μ_r to the mean of the training points assigned to that cluster.

Ideally, the loop must be run until convergence. However, for practical purposes, we run it over a large number of iterations.

It can be proven that the k-Means is guaranteed to converge. Consider the distortion function

$$J(\pi, \mu) = \sum_{i=1}^{n} ||x_i - \mu_{\pi(i)}||^2 \tag{0.1}$$

The cluster assignment step minimizes J with respect to $c(i)$ while holding $\mu_{c(i)}$ fixed. While the centroid updation step minimizes J with respect to μ while holding $c(i)$ fixed. Hence, J must monotonically decrease, and the value of J must converge. It can thus be shown that k-means is exactly coordinate descent on J.

However, J is a non convex function and so, k-means might converge on a local optima (and not the global minima). Hence, the initialisation values of cluster centroid $(\mu_1^{(0)}, \cdots, \mu_k^{(0)})$ matters. In vanilla k-means algorithm, the centroids are initialized by randomly choosing k test examples. A better idea, however, is to follow the "farthest" heuristic. As per this heuristic, we initialize the first centroid randomly, then initialize the second centroid to be the data point farthest away from it. In general, we initialize the jth centroid to be the point whose distance from the preceding $j-1$ centroids is the largest. The k-means++ algorithm is an even better strategy. It selects a point with the probability proportional to the square of its distance from the nearest preceding centroid.

3 Our Implementation

3.1 Pre-processing

We read the image using `matplotlib`'s `imread` function. After that, we reshape the image into a list of points. Each point is represented by a three-feature list (RGB values). Number of points is simply the number of pixels in the image. The k-means function is then called on our image list. This is the code for the same-

```
import matplotlib.pyplot as plt

img = plt.imread('/content/color.jpeg')
plt.imshow(img)
clr_data = img.reshape(img.shape[0]*img.shape[1],img.shape[2])
clr_outputs, clr_centroids = KMeans(clr_data, K=7, num_iters=200)
```

3.2 K-Means Algorithm

We have implemented the vanilla K-Means algorithm described above in abundant detail. Here is our code for the `KMeans` function-

```
def KMeans(data, K, num_iters):

  n = data.shape[0]
  p = data.shape[1]

  centroids = np.zeros((K,p))
  rnd = np.random.randint(0,n-1,size=K)

  for i in range(K):
    centroids[i] = data[rnd[i]]
  outputs = []
  for i in range(num_iters):
    dist = np.empty((n,K))
    for k in range(K):
      dist[:,k] = np.sum((data-centroids[k])**2,axis=1)

    min_centroid = np.argmin(dist,axis=1)

    temp = []
    for k in range(K):
      temp.append([])
    for j in range(n):
      temp[min_centroid[j]].append(data[j])
    for k in range(K):
      temp[k] = np.array(temp[k])
    for k in range(K):
      centroids[k,:] = np.mean(temp[k],axis=0)
    outputs = np.array(temp)
  return outputs, centroids
```

3.3 Image Filter

We applied our colour extraction function to filter out images on the basis of colours.

Given a set of images, our code aims to find images which have a significant amount of a user-specified colour. Firstly, we extract out the major colours from all the given images by applying our KMeans function. We set $K = 4$, so our function extracts out the 4 major clusters in an image. We also need a quantitative value of any RGB colour, so that we can make a comparison between the user-specified colour and the extracted centroids and decide whether they are “close enough”.

For this purpose, we use two functions from the `skimage.color` library, namely `rgb2lab` and `deltaE_cie76`.

The `rgb2lab` function converts the RGB colour space to the CIE 1967 L*a*b* colour space. This is done because the L*a*b* colour space is more useful for predicting small differences in colour. The amount of numerical change in a point in the L*a*b* space has a direct relationship to the amount of perceived difference in colours. The `deltaE_cie76` function simply calculates the Euclidean distance between two points in the L*a*b* colour space.

To decide whether a centroid is “close enough” to a given user-specified colour, we need to set an upper threshold for the Euclidean distance. This value is mostly set through trial and error, since the human notion of colour is not objectively defined, i.e., when we refer to any colour, say blue, we mean a spectrum centred around the “true blue” colour (which includes different shades of blue, like light blue, navy blue, deep blue, etc.). This threshold value determines how large the spectrum is.

If we find that the distance between one of our K centroids and the user-defined colour is less than the threshold, we filter that image as containing the particular colour and output it. Below is our code for the same:

```
flag = False
for k in range(K):
given_color_value = rgb2lab(np.uint8(color_dict[color]))
      found_color_value = rgb2lab(np.uint8(centroids[k]))
      diff = deltaE_cie76(found_color_value, given_color_value)
      if (diff<threshold):
        flag = True
```

4 Applications

- Real-time Colour sensors employ colour identification methods like ours and are an integral part of Cyber-physical systems (CPS) used at various places. For instance, fire detection systems that work on CCTV video feed use colour identification methods.
- To spot sick travellers at airports by feeding thermal images to our algorithm
- To filter images based on colour – an important feature for photo applications like Google Photos and Google Images
- A similar clustering method is also used in a technique called "Color Quantization". Color Quantization has a wide range of varying applications, like image compression.

5 References

- Andrew Ng Lecture Notes [here]
- Visualising K-Means Clustering by Naftali Harris [here]
- MTH552 Lecture Notes [here]