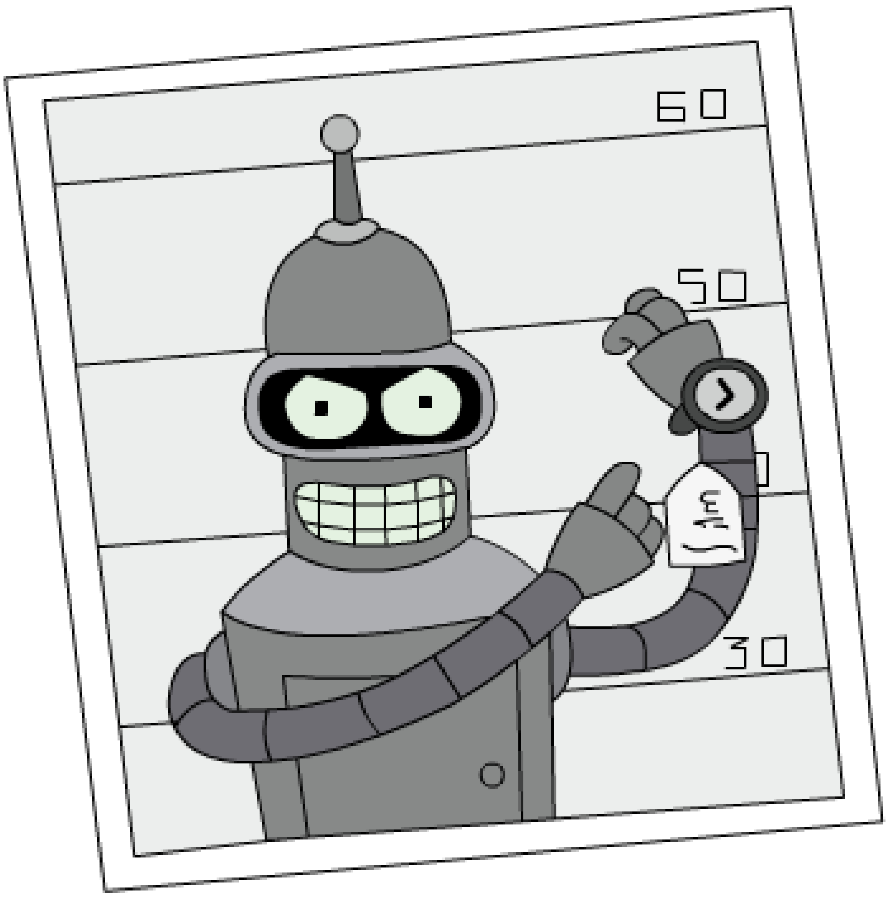
k-Means Clustering

ECE4076/5176: Computer Vision
Lab3 (Weeks 8,9)



Late submission. Late submission of the assignment will incur a penalty of 10% for each day late. That is with one day delay, the maximum mark you can get from the assignment is 9 out of 10, so if you score 9.5, we will (sadly) give you 9. Assignments submitted with more than a week delay will not be assessed. Please apply for special consideration for late submission as soon as possible (*e.g.*, documented serious illness).

Each task in this lab exercise is worth 2% of your final unit grade (a total of 10%). A task is only considered complete if you can demonstrate a working program and show an understanding of the underlying concepts. Note that later tasks should reuse code from earlier tasks.

In this laboratory exercise, you will create a program that clusters pixels in an RGB colour image into k centroid colours using the k-Means Clustering algorithm. In all the tasks below, you may not use any pre-written python libraries for k-Means Clustering, instead, you should write your own code.

References

1. k-means clustering
2. k-means++

Resources mandrill.png, Lab3_kMeans_student_template.ipynb

1 Task1 – Distance Function

In this task, you will build a helper function to compute the squared distance from a set of data points to a set of means (*aka* centroids). You are given a small dataset of five data points $\mathbf{X}$ and two centroids $\mathbf{M}$ to test your function. These are defined as follows

$$\mathbf{X} = \begin{bmatrix} 0.67187976 & 0.44254368 & 0.17900127 \\ 0.55085456 & 0.65891464 & 0.18370379 \\ 0.79861987 & 0.3439561 & 0.68334744 \\ 0.36695437 & 0.15391793 & 0.81100023 \\ 0.22898267 & 0.58062367 & 0.5637733 \end{bmatrix}$$

$$\mathbf{M} = \begin{bmatrix} 0.66441854 & 0.08332493 & 0.54049661 \\ 0.05491067 & 0.94606233 & 0.29515262 \end{bmatrix}$$

If you wrote the function correctly, your answer should be close to:

$$\begin{bmatrix} 0.25977266 & 0.47150141 & 0.10634496 & 0.16664051 & 0.43745224 \\ 0.64767303 & 0.34083498 & 1.0663305 & 0.99096278 & 0.23600355 \end{bmatrix}$$

You will also visualize whether your distances are correct by assigning the data points to the closest centroid (by using different colors) on a 3D scatter plot.

2 Task 2 – k-Means Clustering

In this task, you will implement the k-Means Clustering algorithm. First, you will write a function to generate k random centroids as initialization. Then you will write a function that uses these centroids to perform k-Means Clustering using the following steps:

1. Go through all the pixels in the image and calculate which of the k centroids it is closest to.
2. For each pixel, store the index of the nearest centroid (*e.g.* at the same pixel location in an index 'image'/matrix).
3. Re-compute each centroid by going through all the pixels in the RGB colour image that were assigned to that centroid and taking the mean.
4. Repeat steps 1-3 for a pre-determined number of iterations.

Load the **mandrill.jpg** image and use your function to cluster the pixels of the image into four clusters ($k = 4$). Finally, you will display the results by using the colour of the corresponding centroid assigned to each pixel.
Run the program several times with different random seeds to see if you always converge on the same answer. Try changing the number of centroids from $k = 4$ to other numbers (from 2 to 10) and see how this affects the output and the repeatability of the program.

3 Task 3 – Visualization of Centroids

Looking at the results of the previous tasks 1 and 2, you should be able to observe the clustered output RGB image converging towards 4 mean RGB colour values (for $k = 4$). In this task, you are now asked to write code to visualize the results of the clustering process at every iteration.
For this, you will perform the following steps:

1. First, you will write code to create a 3D scatter plot that shows the data points at their current 'position' along with the current centroids, and additionally display the clustered image (*e.g.* scatter plot and image side-by-side via the use of subplots).

   Hint: The three dimensions of the 3D scatter plot are the RGB channel values. Thus, the RGB colour values from the **mandrill.jpg** image will determine the position of the data points, while you should visualize each data point with a marker in the colour of the centroid it is closest to. Use a different marker types or marker sizes to visualize the centroids to be able to distinguish them from the data points!

2. Then, you will use that function inside your k-Means Clustering implementation to observe how the centroids move with each iteration.

Improving the efficiency:

As it is computationally very expensive to plot all the data points of the image at every iteration, you may increase the efficiency of your implementation by only drawing a total of 250 randomly selected data points.

Improving the interpretability:

To improve the interpretability of what is happening during the clustering procedure, make sure to add a time pause of 1-2 seconds after each iteration so there is enough time for you to observe the convergence of centroids clearly.

4 Task 4 – k-Means++

In this task, you will implement a variant of the k-Means Clustering algorithm called the k-Means++ algorithm (see link provided under References). In the k-Means++ algorithm, only the initialization of the centroids will change, the rest of the implementation is the same as the conventional K-Means Clustering with random initialization. Therefore, first, you will write a function to generate centroids using the k-Means++ initialization by following the below steps:

1. Choose the first centroid randomly from the data points.

2. Compute the distance d between every data point and the most recently selected centroid. Compare d with distance values from previous iterations and record the minimum value as $\mathcal{D}(x)$ for data point x.

3. Compute the probability of selecting each data point x as the next centroid as:

$$\mathcal{P}(x) = \frac{\mathcal{D}(x)}{\sum_x \mathcal{D}(x)}$$

4. Randomly draw a new centroid based on the computed probability distribution. HINT: Compute an array that represents the cumulative distribution from the probability values using the **cumsum** function. Generate a random number r between 0 and 1. The new centroid corresponds to the first element value in the cumulative distribution array that is greater than the value r.

5. Repeat steps 2-4 until all k centroids have been chosen (initialized).

After having chosen k number of centroids, proceed with the standard k-Means algorithm using the selected centroids. Load the **mandrill.jpg** image and use your k-Means++ implementation to cluster the pixels of the image into four clusters ($k = 4$). Finally, you will display the results.

5 Task 5 – Comparison between Random Initialization and k-Means++ Initialization

In this task, you will visualize random initialization and k-Means++ initialization by plotting the initially generated centroids with respect to the final clustering of data points from each method. Also, you will observe the loss of k-Means Clustering and how the two types of initializations affect the convergence of the algorithm.