K-mean with pseudo Huber distance

Mahdi Akbari Zarkesh
9612762638

K-Means Clustering Intuition:

K-means algorithm is an iterative algorithm that tries to partition the dataset into K-pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group. It tries to make the inter-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster.

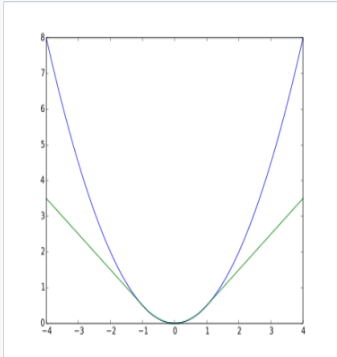
Huber loss function:

The Huber loss function describes the penalty incurred by an estimation procedure f. Huber (1964) defines the loss function piecewise by[1]

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \le \delta, \\ \delta(|a| - \frac{1}{2}\delta), & \text{otherwise.} \end{cases}$$

This function is quadratic for small values of a, and linear for large values, with equal values and slopes of the different sections at the two points where $|a| = \delta$. The variable a often refers to the residuals, that is to the difference between the observed and predicted values $a = y - f(x)$, so the former can be expanded to[2]

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \le \delta, \\ \delta\,|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$



Huber loss (green, $\delta = 1$) and squared error loss (blue) as a function of $y - f(x)$

Pseudo-Huber loss function:

The **Pseudo-Huber loss function** can be used as a smooth approximation of the Huber loss function. It combines the best properties of **L2** squared loss and **L1** absolute loss by being strongly convex when close to the target/minimum and less steep for extreme values. This steepness can be controlled by the δ value. The **Pseudo-Huber loss function** ensures that derivatives are continuous for all degrees. It is defined as[3][4]

$$L_\delta(a) = \delta^2 \left(\sqrt{1 + (a/\delta)^2} - 1\right).$$

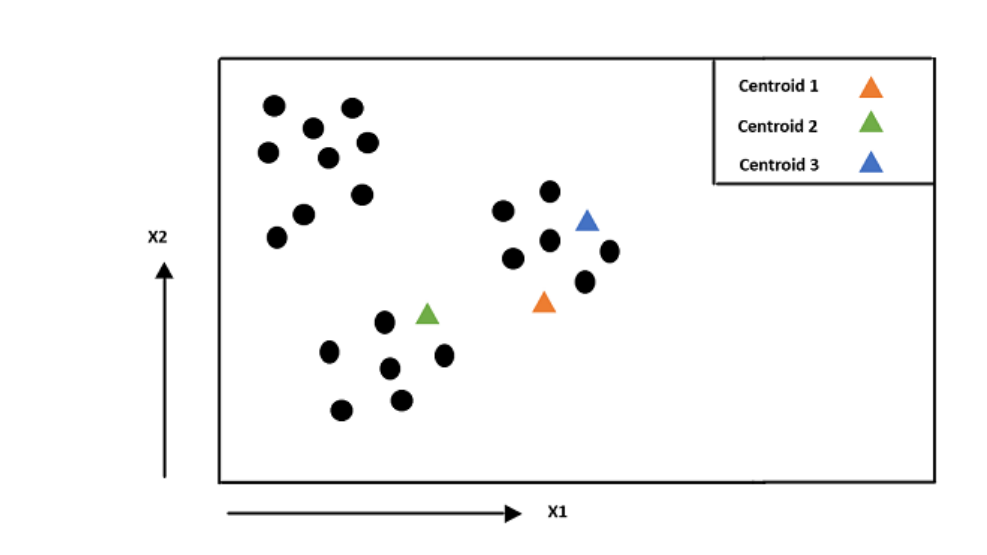
As such, this function approximates $a^2/2$ for small values of a, and approximates a straight line with slope δ for large values of a.

While the above is the most common form, other smooth approximations of the Huber loss function also exist.[5]

Algorithm :

Step 1:

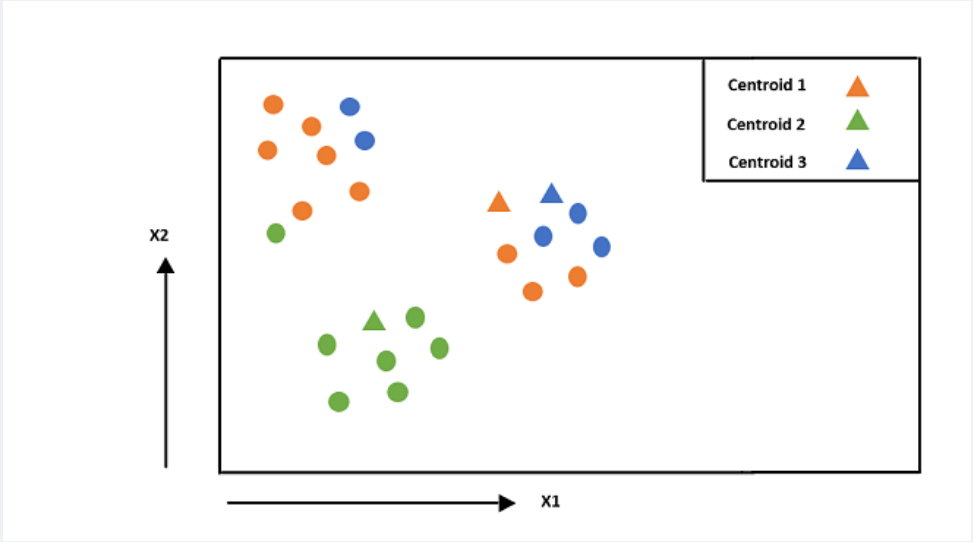
Randomly initialize the cluster centers of each cluster from the data points. In fact, random initialization is not an efficient way to start with, as sometimes it leads to increased numbers of required clustering iterations to reach convergence, a greater overall runtime, and a less-efficient algorithm overall. So there are many techniques to solve this problem like K-means++ etc. Let's also discuss one of the approaches to solve the problem of random initialization later in the article. So let's assume K=3, so we choose randomly 3 data points and assume them as centroids.



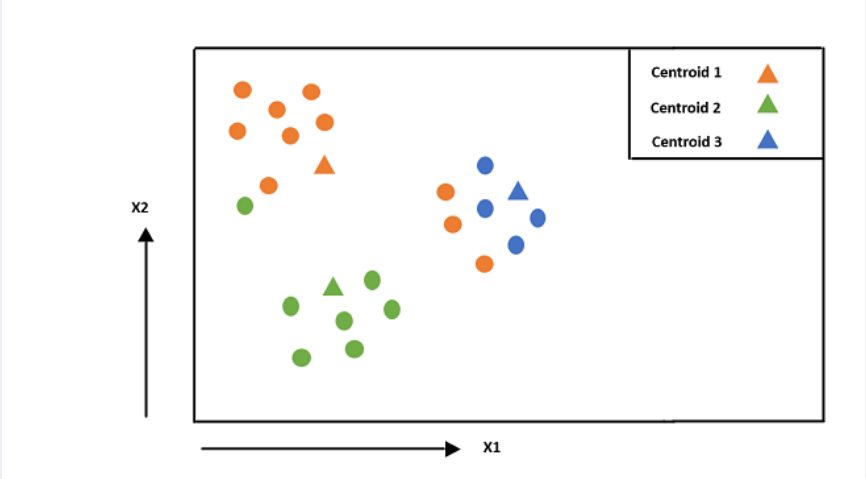
Here three cluster centers or centroids with the green, orange, and blue triangle markers are chosen randomly. Again this is not an efficient method to choose the initial cluster centers.

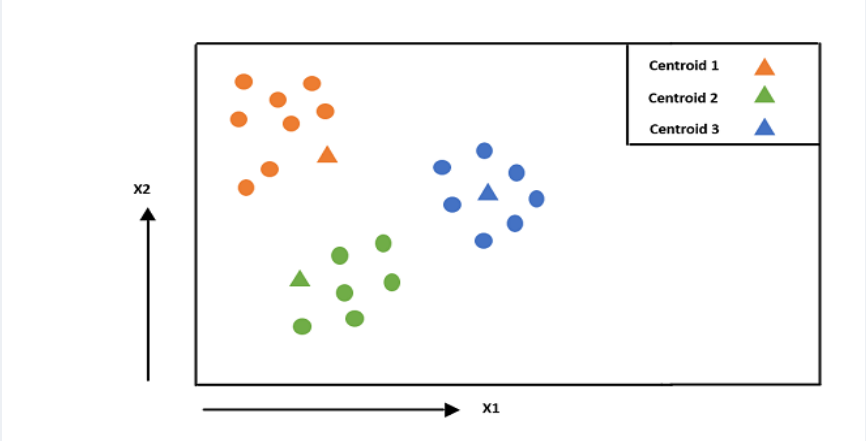
Step 2:
2a. For each data point, compute the euclidian distance from all the centroids (3 in this case) and assign the cluster based on the minimal distance to all the centroids. In our example, we need to take each black dot, compute its euclidian distance from all the centroids (green, orange and blue), and finally color the black dot to the color of the closest centroid.
2b. Adjust the centroid of each cluster by taking the average of all the data points which belong to that cluster on the basis of the computations performed in step 2a. In our example, as we have

assigned all the data points to one of the clusters, we need to calculate the mean of all the individual clusters and move the centroid to calculated mean.
Repeat this process till clusters are well separated or convergence is achieved.



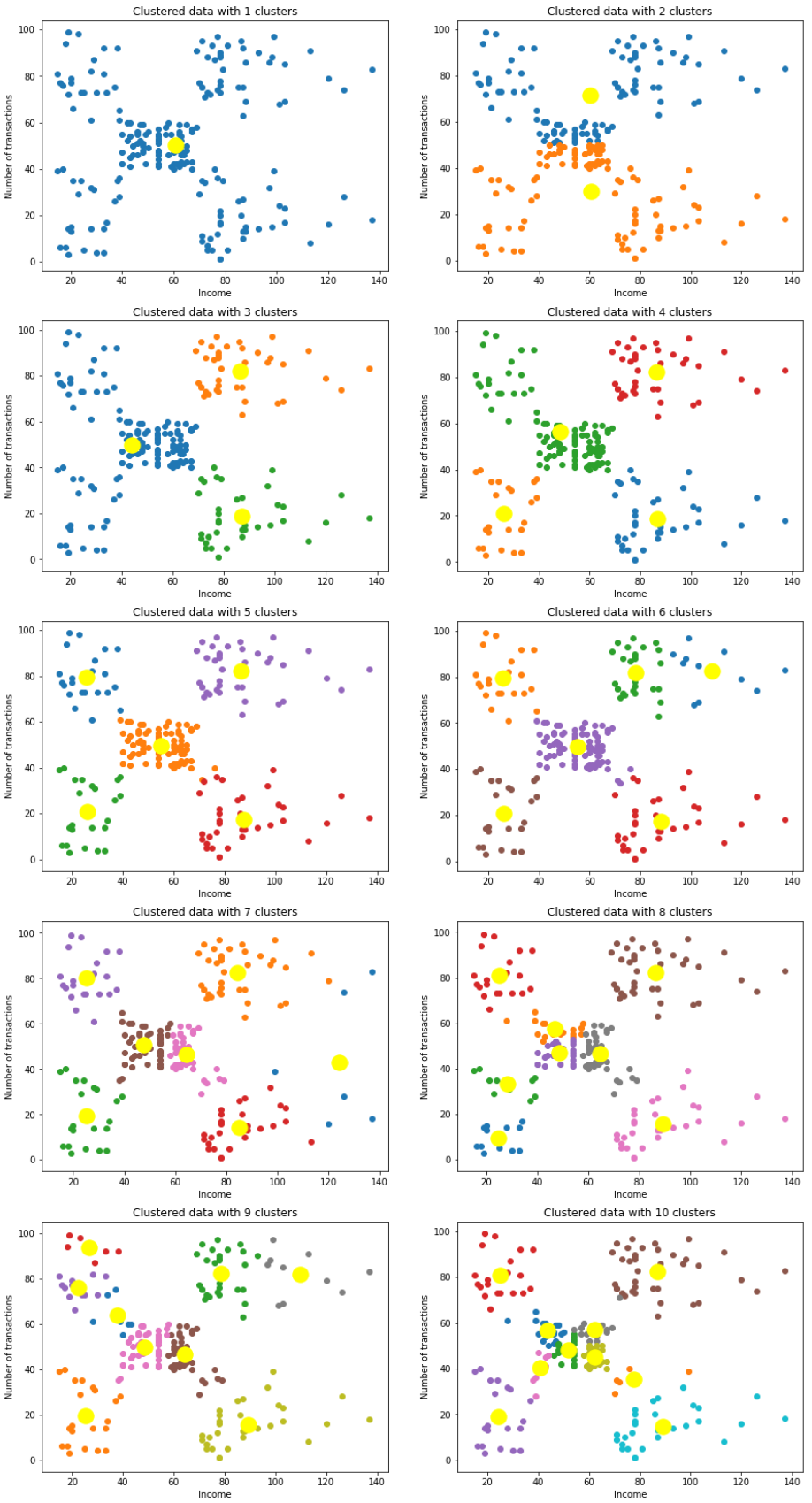
Repeat these steps until convergence is achieved:





Final state

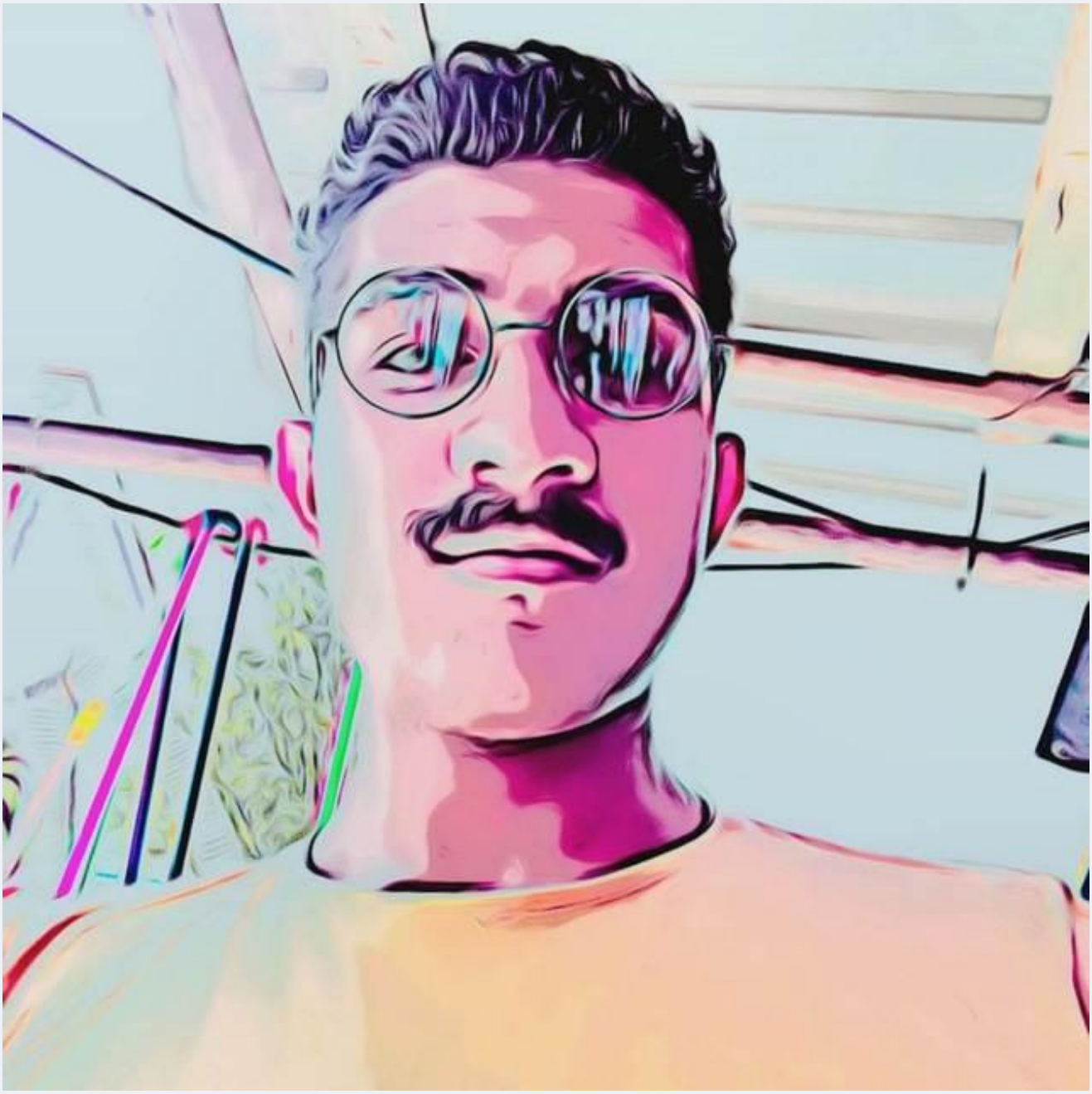
To find out how, let's visualize the clustered data with different clusters starting from 1 to 10 :



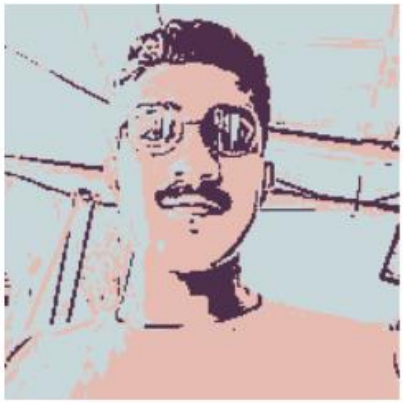
Sample

now look at the our result:

we use photo for data input:



The output of the algorithm:



3- Center

5- Center

As we can see the result doesn't change a lot after we use 10 center but the time of calculate are grows
We repeat this scenario with 50 center and but the result change a little :)

10- Center

20- Center

We can find out with grow the number of center we can reduce the error but From time to time the error reduction percentage is so low that it can be neglected but has a high computational burden on the computer.

30- Center

Thank you :)