K-Means Clustering

k-means clustering is a method of vector quantization, originally from signal processing, that aims to partition **n observations** into **k clusters** in which each observation belongs to the cluster with the nearest **mean** (cluster centers or cluster **centroid**), serving as a prototype of the cluster.

https://en.wikipedia.org/wiki/K-means_clustering

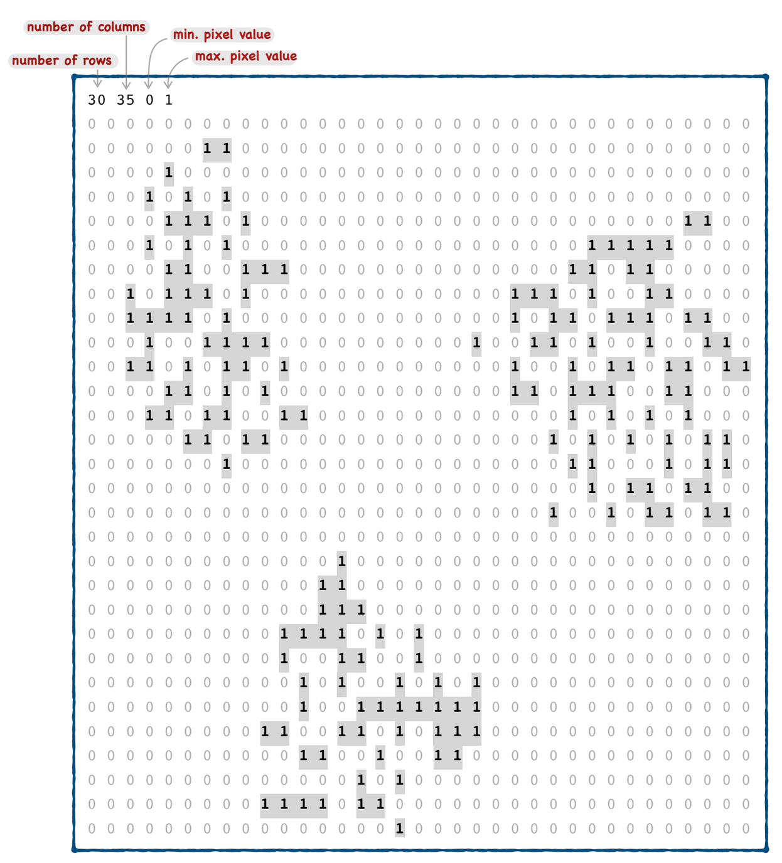
K-means Clustering is a clustering algorithm that works on data that has not been labeled yet. This means that, K-means Clustering is an **Unsupervised Learning** algorithm. K-means simply partitions the given dataset into groups (clusters) where each group has different features. K refers to the total number of clusters existing in the given dataset. The boundaries of each cluster are determined by the euclidean proximity between each of the cluster's members and the cluster's centroid.

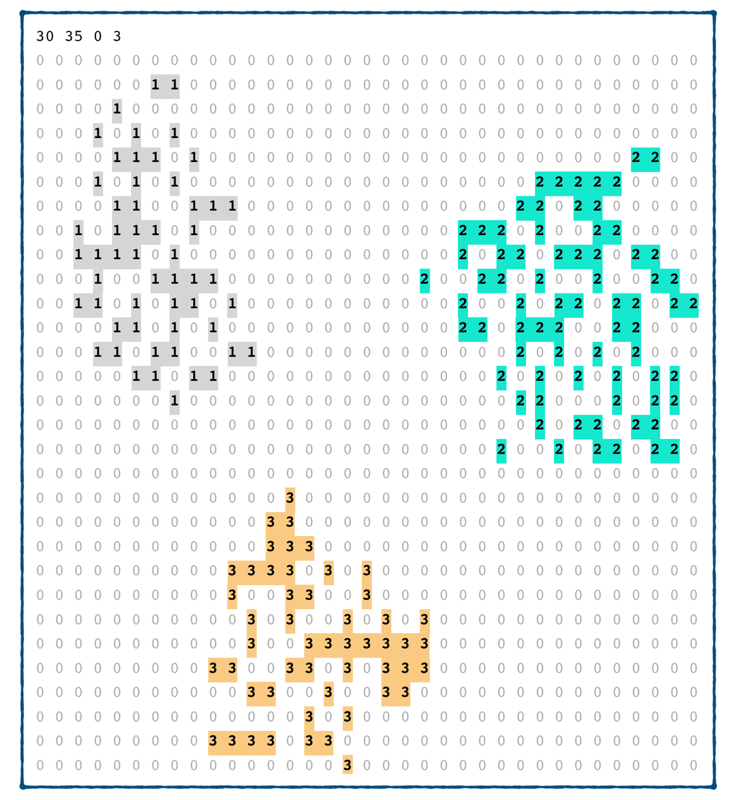
K-means Clustering works best for 2-Dimensional numerical data. In real applications, feature vectors (i.e. data members) usually have multiple dimensions, hence, dimensionality reduction has to be performed before K-means can be applied.

Task:

Given the following binary image (2-dimensions) containing 3 objects, where the background pixels have the value of 0 and the objects the value of 1, detect the objects by homogeneously labeling their constituent pixels. E.g: label ALL pixels belonging to:

object-1 as 1
object-2 as 2
object-3 as 3





Extract points from the binary image. → **1**

PointSet

	x	y
0	x_1	y_1
1	x_2	y_2
2	x_3	y_3
n	x_n	y_n

Cartesian coordinates of each pixel of value >= 1

→ k-mean clustering method **2**

1 - randomly select centroids
2 - reset change
3 - find centroid(s)
4 - find distances to each centroid for each p
5 - find min centroid distance's label
6 - relabel the given point
7 - do step 4 to 6 for all points
8 - repeat 2 to 7 until no more changes

	x	y	cl
0	x_1	y_1	l_i
1	x_2	y_2	l_i
2	x_3	y_3	l_i
	where i is 1...k; in our example, i will end up being either 1, 2 or 3		
n	x_n	y_n	l_i

→ Display the labeled points **3**

Extract the x,y-coordinates of ALL the points belonging to objects (i.e. pixels=1) from the above binary image into an output.txt file.

extracting points

Algorithm **1**

Algorithm - steps for extracting points from a binary image:

- step-0: infile ← open input.txt image file (and all other initializations)
- step-1: read the image L→R & T→B — i.e. left to right , top to bottom
- step-2: p(x,y) ← read point from image — i.e. point (pixel) **p** coordinates x, y
  if(p(x,y).value >= 1)
  output x , y. — i.e. output x,y coordinates to PointSet out-file
- step-3: repeat steps 1 to 2 until eof — i.e. eof means 'end of file'

Algorithm **2**

k-mean clustering

Algorithm steps for k-mean clustering:

step-0: instantiations (allocation), initializations

pointSet ← open input image file and extract all points
n ← number of points (count the points when extracting)
k ← is given by the user
change ← 0

2 alternative approaches:

1. define "change" based on the Centroids travel distance. If the Centroids travel below some small threshold (maybe another hyper-parameter), then there is no change and we are done!
2. define "change" as the variation in the 'total distance' from all labels (in a cluster) to their Centroid. The smallest total distance wins. This means that, the final state of each run, i.e., the labeled results as well as their corresponding 'total distance' will have to be stored.

$\mathbf{cl_1, cl_2, \ldots, cl_k}$ ← clusters
$\mathbf{c_1, c_2, \ldots, c_k}$ ← centroids

step-1: **randomly** select **k** <u>different</u> points from among the points $\mathbf{p}(x_i,y_j)$ in the **pointSet** and appoint them as starting **centroids**., where $1 \le i,j \le n$

$c_1, c_2, \ldots, c_k$ ← some random $\mathbf{p}(x_i,y_j)$

go to step-4

step-2: Reset changes
change ← 0

step-3: Find centroids
for each cluster $\mathbf{cl_{1\ldots k}}$ compute **centroid** c_i **(** $x(cl_i)$, $y(cl_i)$ **)**, where $1 \le i \le n$.

sum of ALL x-coordinates of points labeled as e.g. 1 — sum of ALL y-coordinates of points labeled as e.g. 1

$$x(\mathbf{cl_i}) = \frac{\sum_i x_i}{\sum_i 1} \qquad y(\mathbf{cl_i}) = \frac{\sum_i y_i}{\sum_i 1}$$

e.g. 1; number of points in cluster i — e.g. total number of points labeled as 1

mean (average) = $\frac{\text{sum of values}}{\text{\#of values}}$

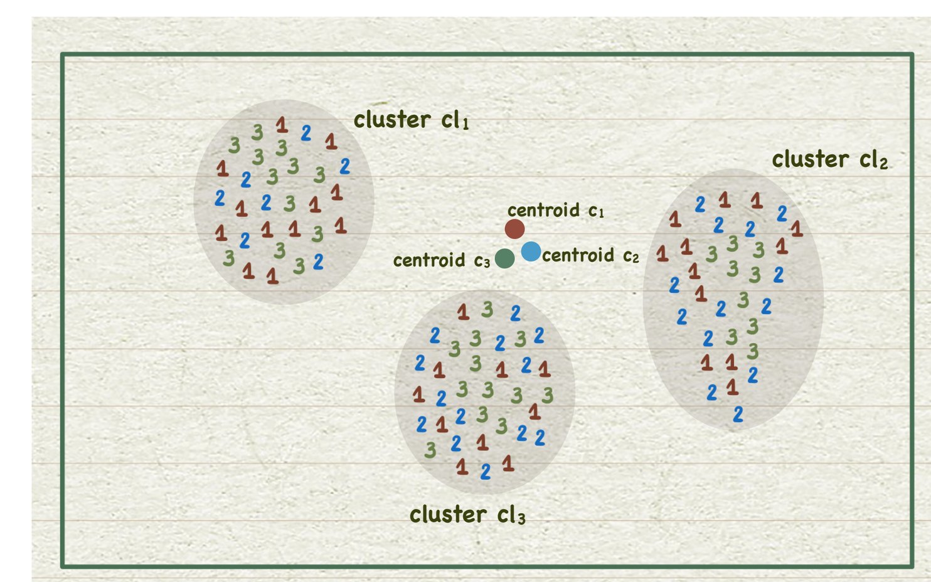
As centroids reposition (per iteration), this number changes.

step3.1 cl_i - get a cluster from among the clusters
step3.2 compute centroid c_i for cl_i
step3.3 repeat steps 3.1 to 3.2 until all clusters are precessed

e.g. centroid $\mathbf{c_1}$ ($x(\mathbf{cl_1})$, $y(\mathbf{cl_1})$) think: the Center of Mass of ALL points labeled 1
if more dimensions, (... , Z(cli))

Def.

Centroid: (the center of mass) in mathematics and physics, the centroid or geometric center of a plane figure is the arithmetic mean position of all the points in the figure. Informally, it is the point at which a cutout of the shape could be perfectly balanced on the tip of a pin.



extra dimensions

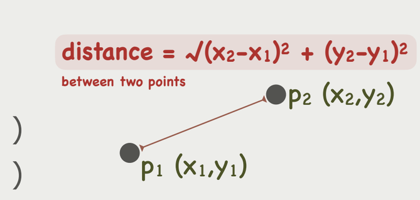
Note that: our example computes only in 2-dimensions x and y. If we want to increase the number of dimensions (e.g. to 7-dimensions because our feature vector has 7 attributes), we must compute the centroid for each of those extra dimensions i.e. z(cl_i) , k(cl_i) , m(cl_i) ,... . Also we extend the distance calculation by those same extra dimensions i.e.

$$\text{distance} = \sqrt{\ldots + (z_2-z_1)^2 + (k_2-k_1)^2 + (m_2-m_1)^2}$$

step-4: For each point **p** in **pointSet**, **find distance** from **p** to each centroid $\mathbf{c_i}$

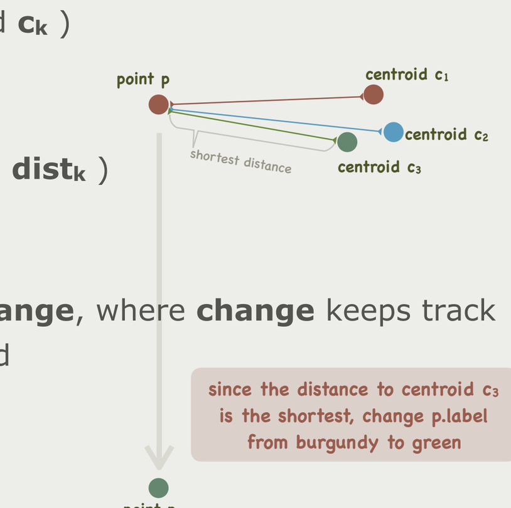
p ← get a point from among **pointSet**

compute **k** distances
$\mathbf{dist_1}$(distance from **p** to centroid $\mathbf{c_1}$)
$\mathbf{dist_2}$(distance from **p** to centroid $\mathbf{c_2}$)
⋮
$\mathbf{dist_k}$(distance from **p** to centroid $\mathbf{c_k}$)



step-5: Find min distance's label

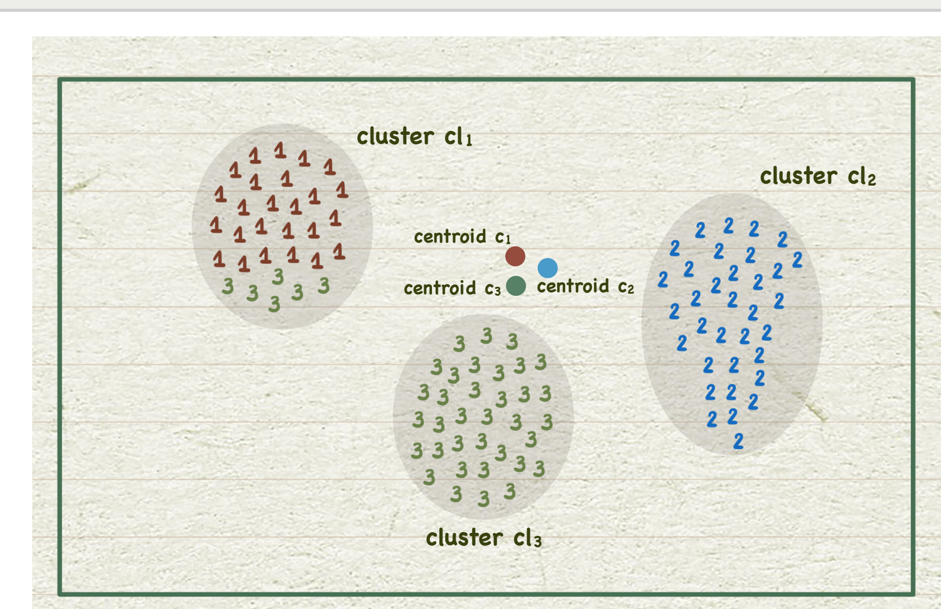
minLbl ← **min**($\mathbf{dist_1}$, $\mathbf{dist_2}$,..., $\mathbf{dist_k}$)



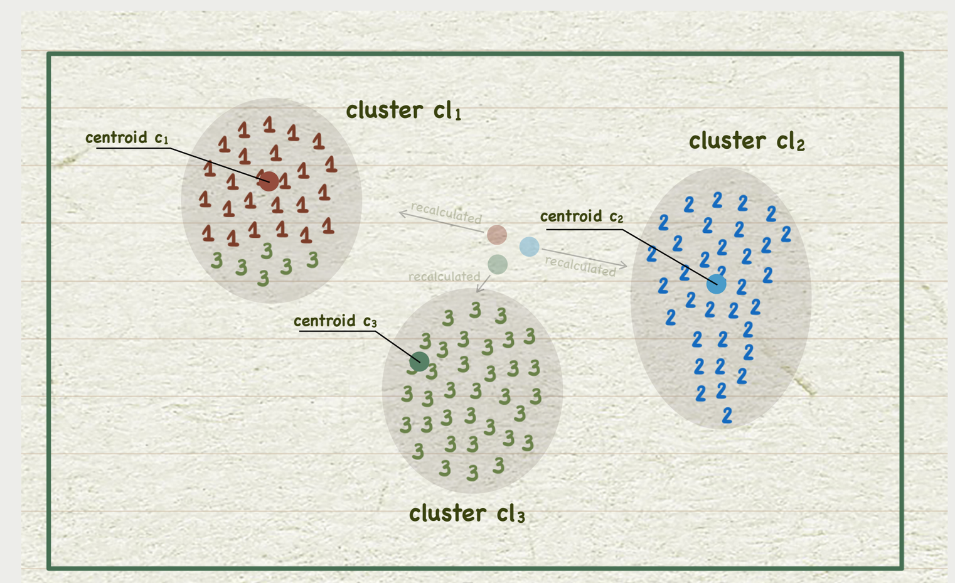
step-6: Relabel point **p** and update the total **change**, where **change** keeps track of how many points have been relabeled

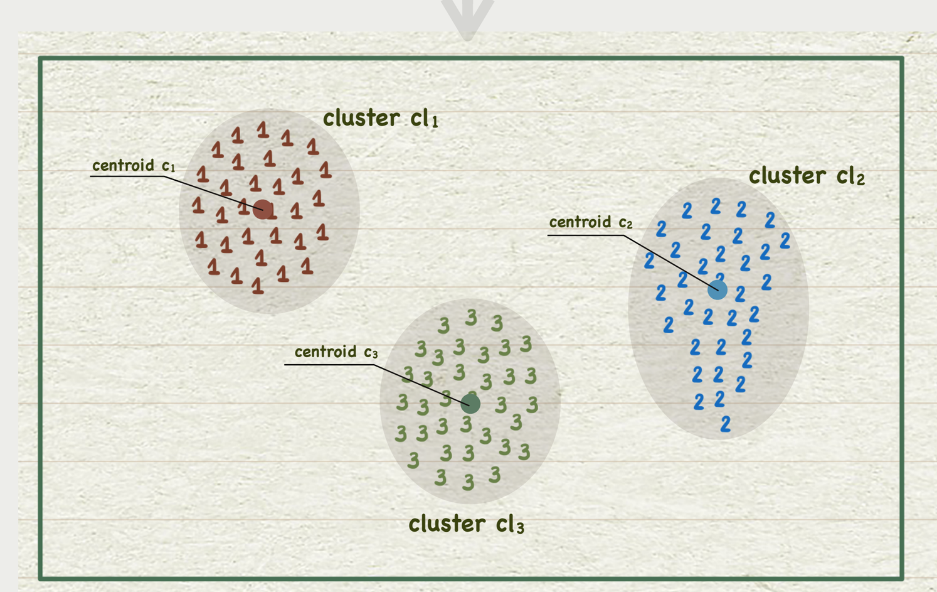
if(**p.label** ≠ **minLbl**)
p.label ← **minLbl**
change++

step-7: Repeat step-4 to step-6 until ALL **n** points have been processed



step-8: repeat step-2 to step-7 until NO more changes are possible or change threshold is below min.





Sometimes there will be an oscillating point that toggles e.g. blue to burgundy to blue. Those points are very rare, increasing the precision of distance measuring (e.g. float → double) should fix that.

standard deviation σ is a measure of how spread out the numbers are.

$\sigma = \sqrt{\text{variance}}$

variance is the average of the squared differences from the mean.

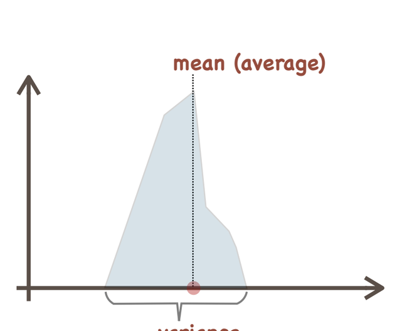
steps for computing variance:

Step1 work out the Mean (the simple average of the numbers)
Step2 n_i ← get a number_i
result_i ← $(n_i - \text{Mean})^2$ // calculate the squared difference
Step3 repeat step 2 until all numbers are processed
Step4 compute variance:

$$\text{variance} = \frac{\text{result}_1 + \text{result}_2 + \ldots + \text{result}_i}{i}$$

In probability theory and statistics, **variance** is the expectation of the squared deviation of a random variable from its mean. Informally, it measures how far a set of (random) numbers are spread out from their average value.

scattered → large variance
compact → small variance



What if we do NOT know k?

- in step0 k (unknown) ← start with 1
- in step7 compute **variance** for each k-cluster then sum-up
- in step8 repeat steps 2 to 7 until variance is acceptable (to the user), otherwise k++.