50.1 What is Clustering?

So far we have seen classification and regression problems. For a given dataset D = {x_i, y_i}

a) If $y_i \in \{0,1\}$, then we call it a 2-class classification
b) If $y_i \in R$, then we call it a regression.

In the case of both classification and regression, we are trying to find a function that is used to predict 'y_i', when 'x_i' is given as the input. In case of clustering we are given a set of points D = {x_i} and there are no y_i's. The task in clustering is to group similar points.

Definition

Clustering is the task of grouping a set of objects in such a way that objects in the same group are more similar to each other than those in other groups.

Each group here is called a cluster. The points in the same cluster are close together. The points in different clusters are far away from each other. The task in clustering is grouping the similar points.

There are 3 techniques in clustering. They are

a) K-Means Clustering
b) Hierarchical Clustering
c) DBSCAN

We perform clustering when we want to group similar items in our dataset based on our definition of similarity. Since clustering doesn't have the class labels (or) the ground truth, it is hard to measure if a clustering is good or bad in a very rigorous and critical way. It all depends on the problem and the context we are working in. Clustering just like classification is very dependent on the features used. The clustering output can sometimes help us come up with new features.

50.2 Unsupervised Learning

So far we have seen Regression and Classification problems, and these two types of problems come under Supervised Learning.

In supervised learning, **f(x) = y**. Here ‘y’ supervises us in determining the underlying function f(x).

The dataset for supervised learning is represented as $D = \{x_i, y_i\}_{i=1}^n$, whereas in unsupervised learning, we do not have **y_i’s**. The dataset for unsupervised learning is represented as $D = \{x_i\}_{i=1}^n$.

In semi-supervised learning, we have a majority of the points without labels and a very few points with the labels. This happens when the cost of labelling is computationally expensive.

Here for a few points, we have the values for ‘y_i’, and for most of the points, we do not have the values for ‘y_i’. This is in between supervised and unsupervised, and hence we call it **semi-supervised**.

50.3 Applications

There are tons of applications of clustering. The major ones are

a) **E-Commerce**

In E-commerce, companies group similar customers based on their purchasing behaviour. Here the similarity is based upon the type of products they purchase, the type of the debit/credit cards they use, their geo location, etc. Once these customers are grouped into different clusters, then depending on their purchasing habits, different deals, discounts are offered.

b) **Image Segmentation (Computer Vision and Image Processing)**

Here we first cluster/group the image pixels based on their similarities and after that we apply ML algorithms to perform object detection.

In cases where manual labelling is time consuming and expensive, clustering can be used. For example, in the Amazon Fine Food reviews dataset, someone has already reviewed each of the reviews and labelled them as positive or negative. In case, if we are not given the class labels, then we could cluster the given data points into groups.

50.4 Metrics for Clustering

For clustering, the dataset is denoted as **$D = \{x_i\}$**. So far in the classification and the regression tasks, we had **y_i's**, but here in clustering, we do not have **y_i's**. All the performance metrics in Classification and Regression need **y_i's**. So as all the performance metrics known to us so far require the 'y_i' values, those metrics do not work in clustering.

If a group of points lie in the same cluster, we call it **Intra-cluster**, and if a group of points are spread across different clusters, then we call it **Inter-cluster**. All the points from the given data set are grouped in such a way that the intra-cluster distances are small and inter-cluster distances are high.

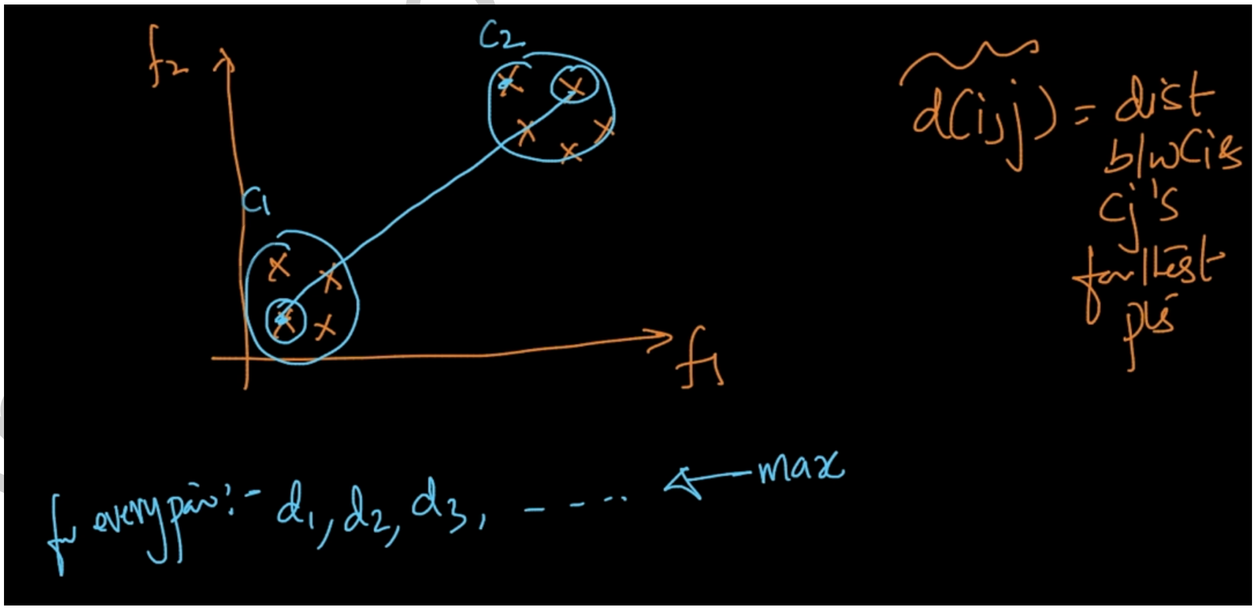
The core idea/basis of the measure of clustering is the intra-cluster distance has to be low and the inter-cluster distance has to be high. There is one metric called **Dunn Index**, which is denoted by 'D' and is given as

$D = \min_{i,j} \text{distance}(i,j)/\max_k \text{distance}^{|}(k)$

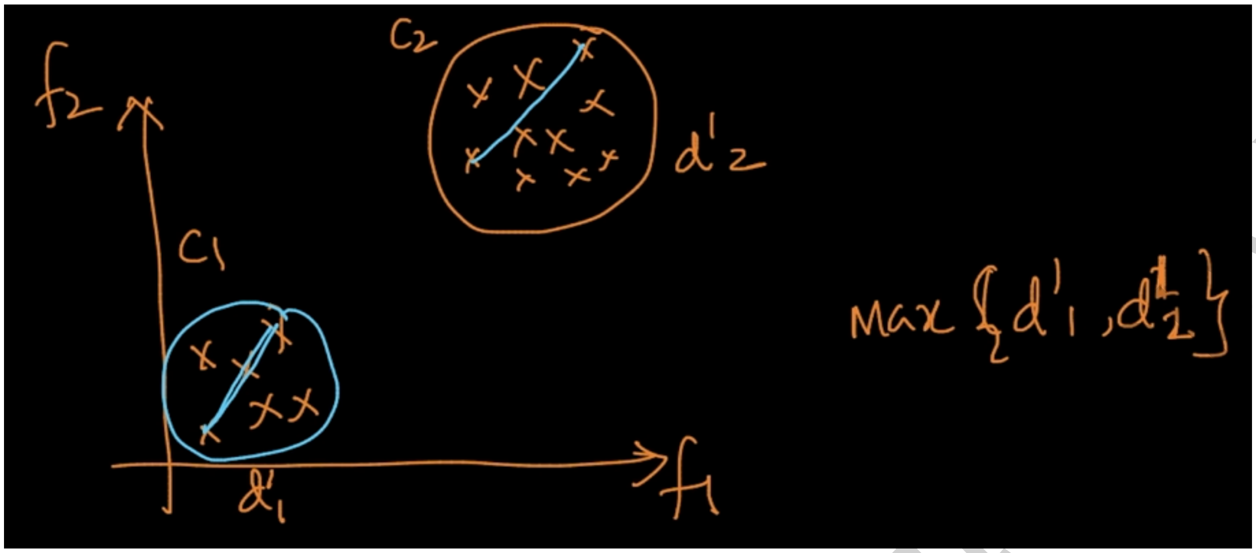
Where distance(i,j) → distance between the farthest points of the clusters 'C_i' and 'C_j'

$\text{distance}^{|}(k)$ → Intra-Cluster distance

If 'D' is high, it implies good clustering. For every pair of points from 'C_i' and 'C_j', we have to compute **distance(i,j)**



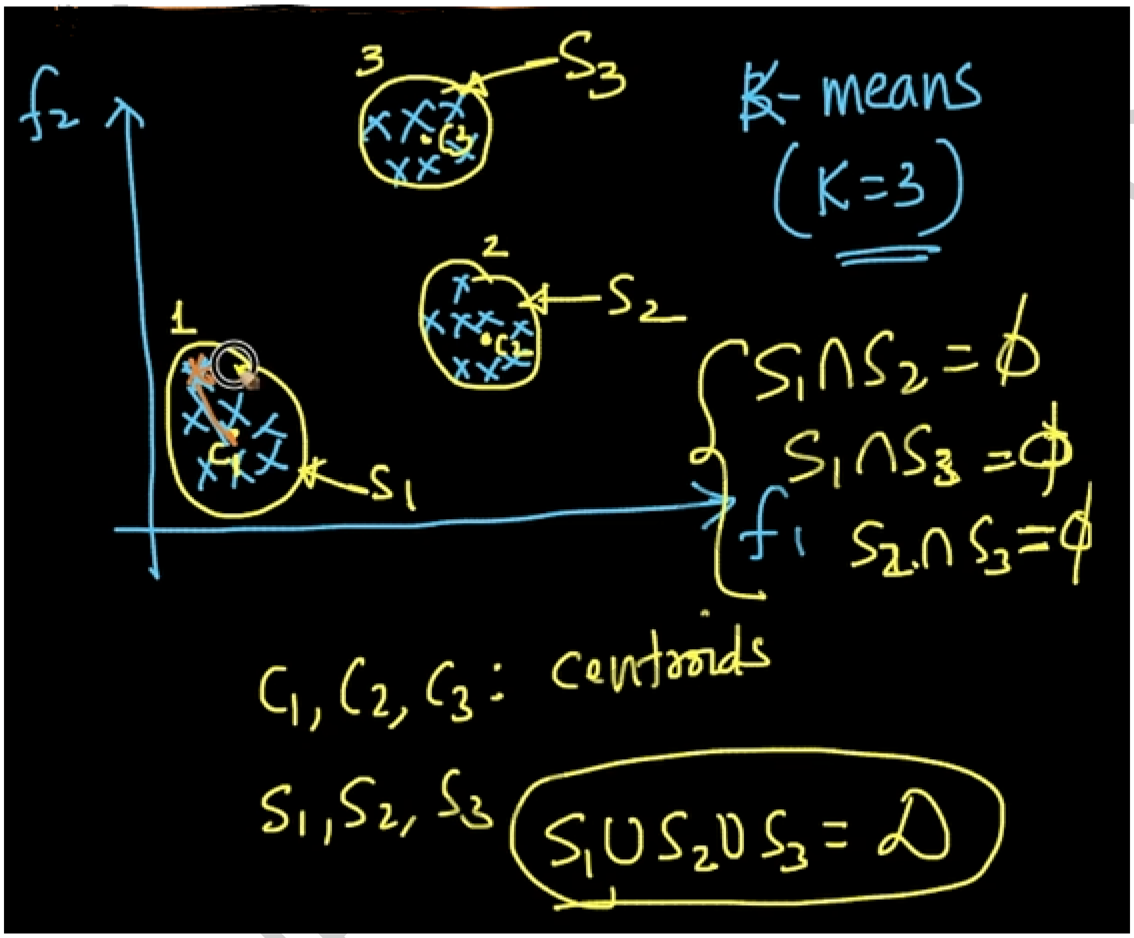
f_2
f_1
C_1
C_2
$d(i,j)$ = dist b/w C_i's C_j's farthest pts
for every pair:- $d_1, d_2, d_3, \ldots$ ← max



For ideal clusters, the value of the Dunn Index should be high, and for that the distances between the points in the same cluster should be as much small as possible, and the distance between the different clusters should be as much large as possible.

50.5 K-Means: Geometric Intuition, Centroids

K-Means clustering is one of the popular and the simplest clustering algorithms. The value ‘K’ in the K-means algorithm denotes the number of clusters.



Let us assume we are given a 2-Dimensional dataset ‘D’ with number of clusters (K)=3. So the number of centroids is also equal to 3.
Let ‘S_1’, ‘S_2’, ‘S_3’ be different sets of elements, and ‘C_1’, ‘C_2’ and ‘C_3’ be their respective centroids.
$S_1 \cup S_2 \cup S_3 = D$
$S_1 \cap S_2 \cap S_3 = \Phi$
$S_1 \cap S_2 = \Phi$, $S_1 \cap S_3 = \Phi$, $S_2 \cap S_3 = \Phi$
Here all the data points belong to one or the other set and no data point exists in more than one set.

Number of Sets = K (ie., S_1, S_2, S_3, ….., S_k)
Number of Clusters= K (ie., C_1, C_2, C_3, ….., C_k)

For any set ‘S_i’, its centroid is given as **$C_i = (1/n)*\Sigma_{x_j \in S_i} x_j$**

K-means clustering is a centroid based clustering scheme. Every point is assigned to the cluster closer to it.

The core idea of K-means clustering is to find ‘K’ centroids and each point is assigned to the cluster whose centroid is nearer to it. The biggest challenge is to find these 'K' centroids. There are algorithms to find out these 'K' centroids and one of the most commonly used algorithms is **Lloyd's algorithm**.

50.6 K-Means: Mathematical Formulation: Objective Function

So far we were given a dataset ‘D’ of ‘n’ data points and our job is to find the ‘K’ centroids.

$D = \{x_i\}_{i=1}^n$

The ‘K’ sets are ‘S_1’, ‘S_2’, ‘S_3’, …., ‘S_k’

The ‘K’ centroids are ‘C_1’, ‘C_2’, ‘C_3’, …., ‘C_k’

The objective function for K-means algorithm is given as
$\textbf{arg-min}_{C1,C2,C3,..,Ck}\ \boldsymbol{\Sigma_{i=1}^{k}\, \Sigma_{x\in Si}\ ||x\text{-}C_i||^2}$ such that $x \in S_i$ and $S_i \cap S_j = \varnothing$

We have to find the cluster centroids such that the points belonging to the respective clusters should be as much nearer as possible to these centroids, so that the intra-cluster distance is minimized.

This optimization problem is very hard to solve from the point of computation. In such cases, we go with the approximation algorithms, and find out the approximate solution for this problem, but not the exact solution, using a few hacks. One such approximation algorithm is **Lloyd’s algorithm**. It is a very simple and a good approximation algorithm.

50.7 K-Means Algorithm (Lloyd's Algorithm)

1) **Initialization**

   From the given dataset 'D', we have to pick 'K' points randomly, and assume them to be the centroids. Let us denote them as C_1, C_2, C_3, …, C_k.

2) **Assignment**

   For each point 'x_i' in the dataset 'D', we have to compute the distance of each of the above 'K' centroids from this point, and pick the nearest centroid. Let us denote this nearest centroid as 'C_j'.

   Add the point 'x_i' to the set 'S_i'(which is associated with the centroid 'C_j').

3) **Recompute Centroid (Update Stage)**

   Recalculate/update 'C_j' as follows

   $Cj = (1/|S_j|)*\Sigma_{xi \in Sj}\ x_i$

4) Repeat the steps 2 and 3 until convergence. Here convergence is the stage where the centroids do not change much.

   For example, at the end of stage 2, if the centroids are {C_1, C_2, C_3, …., C_k} and at the end of stage 3, if the centroids are {$C_1^|$, $C_2^|$, $C_3^|$, ….., $C_k^|$}, then during convergence the distance between the old and the new centroids is very small.

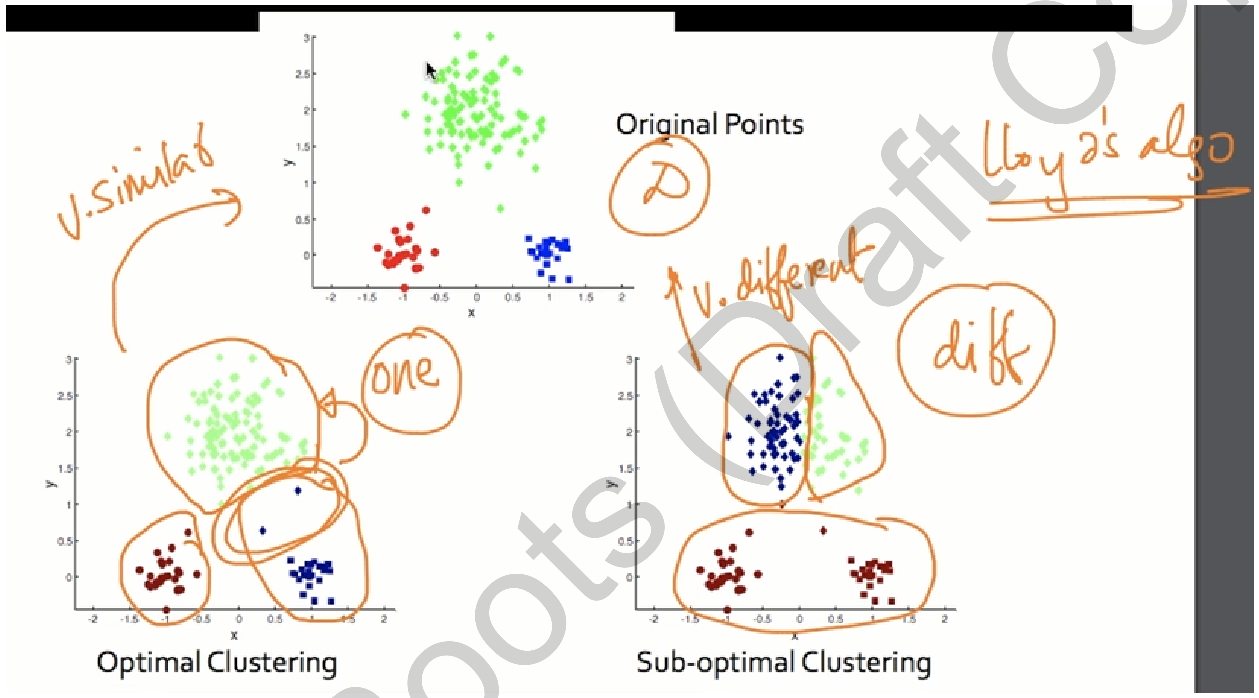
   (ie., C_1-$C_1^|$, C_2-$C_2^|$, C_3-$C_3^|$, …, C_k-$C_k^|$ has to be very small)

   Now finally the centroids we get are C_1, C_2, C_3, …., C_k and the final sets/clusters of points are S_1, S_2, S_3, …., S_k.

50.8 How to initialize: KMeans++

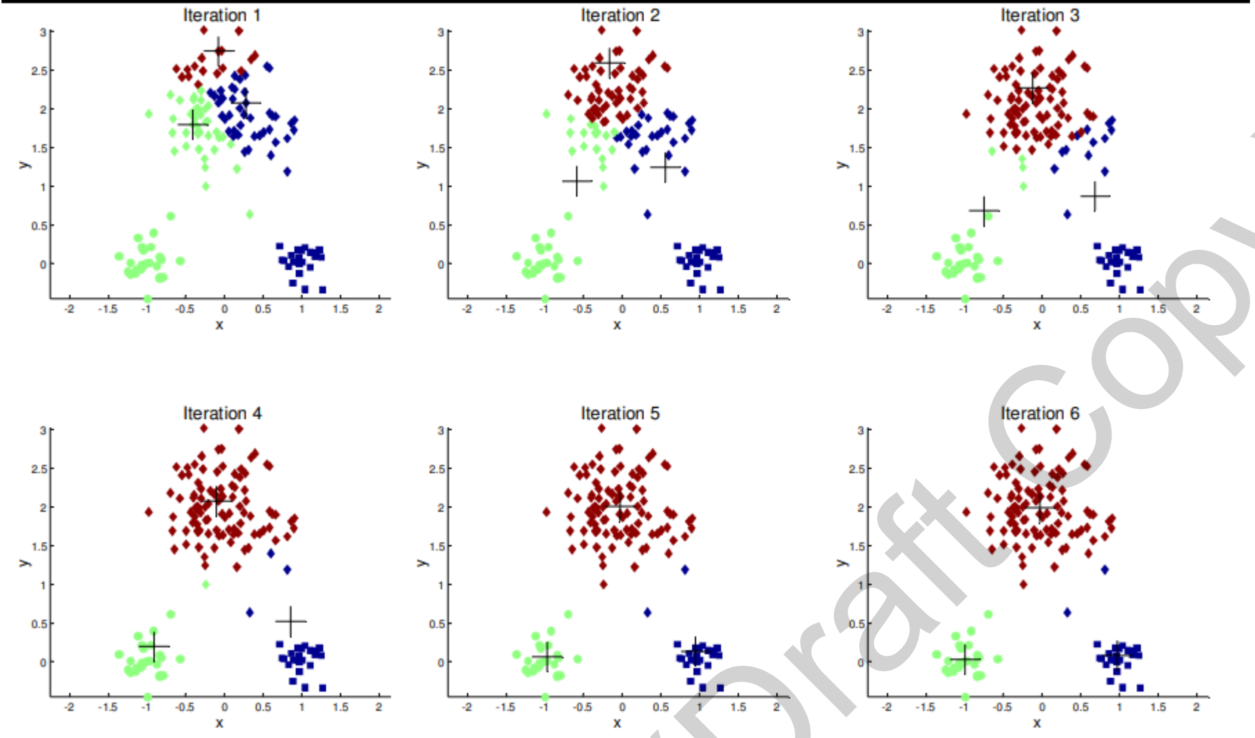
Lloyd’s algorithm has initialization as the first stage. Here we pick ‘K’ points randomly as the centroids. This process of picking the points randomly as the centroids is called **Random Initialization**. But there is a problem with K-means clustering called **Initialization Sensitivity**.

Initialization Sensitivity means the final clusters and centroids depend on how randomly we pick the points as centroids during the initialization.

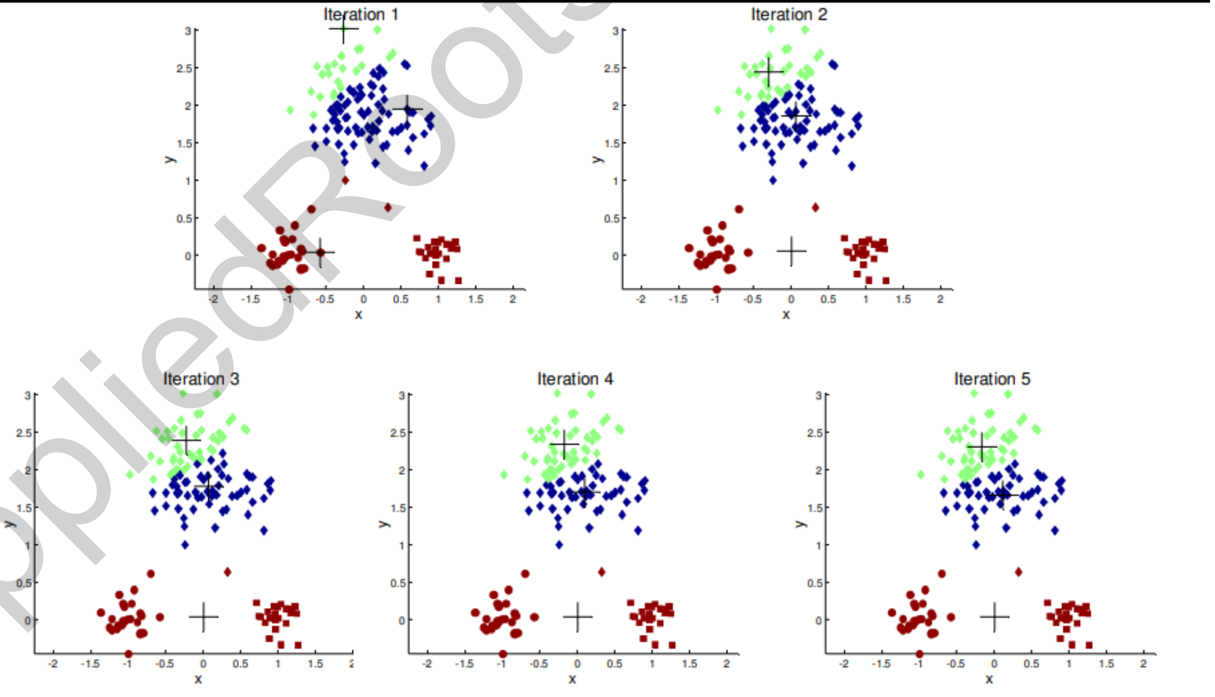


When we look at the points in the above given 3 plots, the plot with the name “Original points” represents the points assigned to clusters manually. The plot with the name “Optimal Clustering” represents the perfect formation of clusters, and the plot “Sub-optimal clustering” represents another formation of clusters which are incorrect. So here the points are the same, and they remain in their positions, but the only difference lies in the clusters they are assigned to. This kind of incorrect clustering happens due to random initializations of the centroids in K-Means clustering.

Differences in initialization of the centroids results in differences in clustering. Let us look at below how one initialization has formed the clusters



Here the points are clustered perfectly. Whereas if there is a slight change in the initialization of the centroids, then for every iteration the clusters chang as shown below



This change in the clusterings of the points is just because of randomization in the initialization of the centroids.

How to deal with Initialization Strategy?

Method 1:

Repeat the K-means approach with different initializations. Pick the best clustering based on smaller intra-cluster and large inter-cluster distances

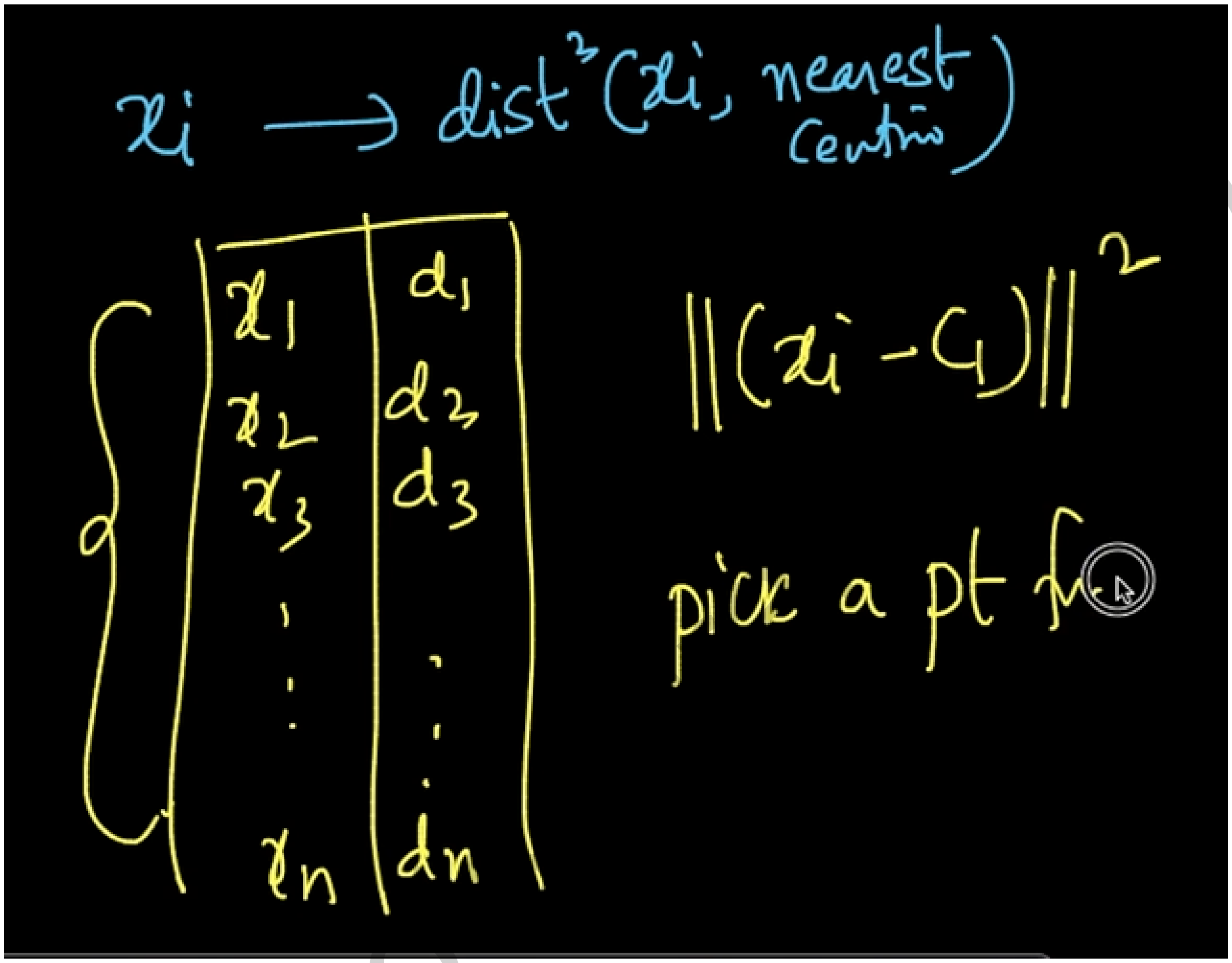
(or)

Method 2: KMeans++

Instead of using random initialization like in KMeans, here the random initialization is replaced by smart initialization.

Initialization in KMeans

1) Pick the first centroid randomly from the dataset 'D'. Let it be 'C_1'.
2) For each $x_i \in D$, create a distribution as follows
   For each 'x_i', compute the square of its distance to the nearest centroid. (ie., for every $\mathbf{x_i \rightarrow distance^2(x_i,}$ **nearest centroid)**). Out of all the points, we have to keep the centroid 'C_1' aside and compute the square of the distances of the remaining points to 'C_1'. (ie., $||x_i-C_1||^2$)



Pick a point from D-{C_1} with probability proportional to 'd_i'. It means if 'd_i' is large for any point, then the chance of picking that point as the next centroid is large.

So the points that are far away from 'C_1' have a high probability of getting selected as the next centroid and it is appropriate. We are trying to pick the points that are as far as possible from the centroids that are already picked up.

We also can go with the deterministic way of picking the points as centroids, where the point with the highest value of **distance^2(x_i, C_1)** is picked. But we don't do it because if there are outliers in our data at far distances, then their value of **distance^2(x_i, C_1)** will be the highest and will be chosen as centroid, which could not be accepted.

KMeans++ gets affected by the outliers severely if a deterministic approach is followed, whereas it gets affected moderately by the outliers if a probabilistic approach is followed. In a deterministic approach, outliers get selected as centroids. So it is better to go with the probabilistic approach to reduce the impact of the outliers.

The probabilistic approach is not always guaranteed to avoid outliers as an extreme outlier has a higher probability of being picked up as a second centroid. It just minimizes the probability of picking the outliers slightly, but not significantly. If we are much concerned about outliers in the data, it is best to use **Local Outlier Factor (LOF)** as a preprocessing step.

The only difference between KMeans and KMeans++ is the initialization strategy. KMeans++ is a probabilistic approach, and no one uses the deterministic approach now. KMeans++ is better than KMeans in most of the cases.

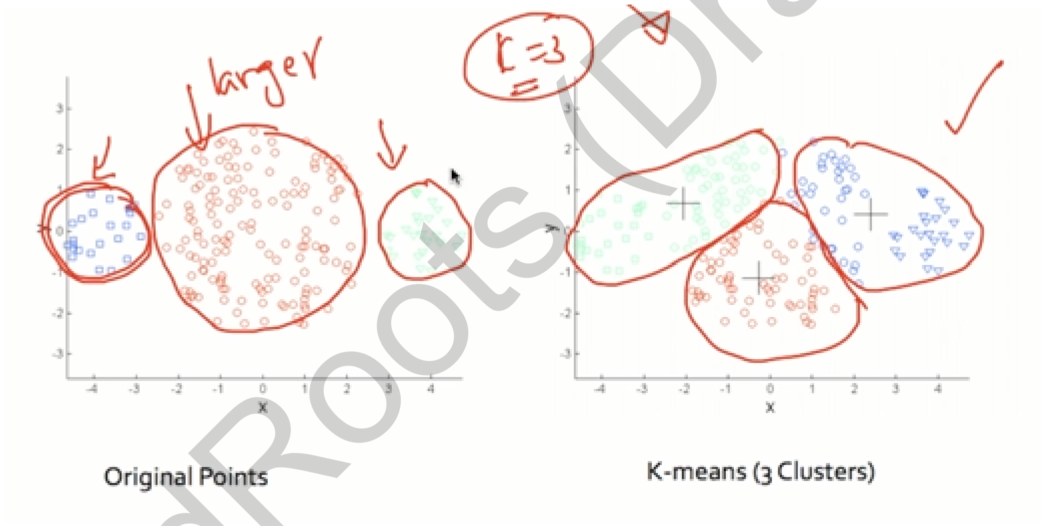
50.9 Failure Cases/Limitations

1) K-Means doesn't work well when the clusters are of

a) Different Sizes

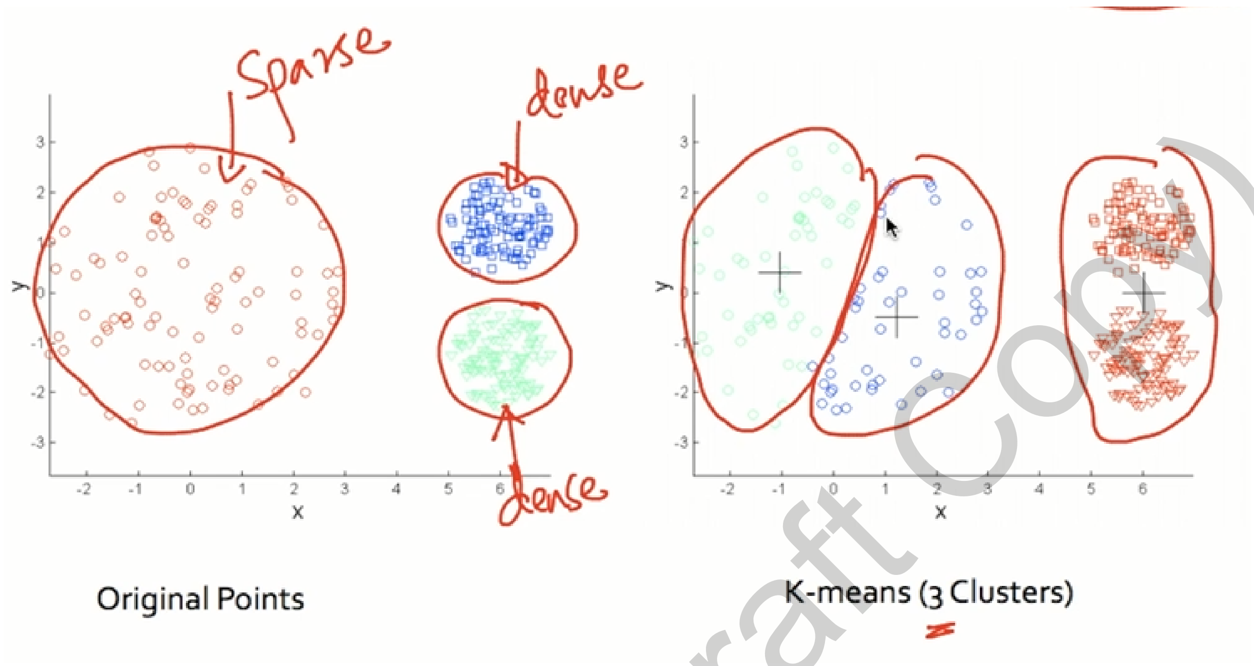
K-means tends to prefer creating clusters that are roughly of the same size. K-means tends to look more for the intra-cluster distances, than the inter-cluster distances.

K-means splits the points into clusters of almost the same size. If the initial given clusters are of different sizes, then the clustering goes wrong. Moreover this problem arises when the initial clusters of different sizes are closer to each other. It doesn't arise when these different sized clusters are far away from each other. Here the 'size' refers to the cardinal area in the space.



b) Different Densities

When we have clusters with high density, then K-means try to stretch those clusters and forms clusters of roughly the same size and less density



c) Non Globular (or) Non Convex shapes

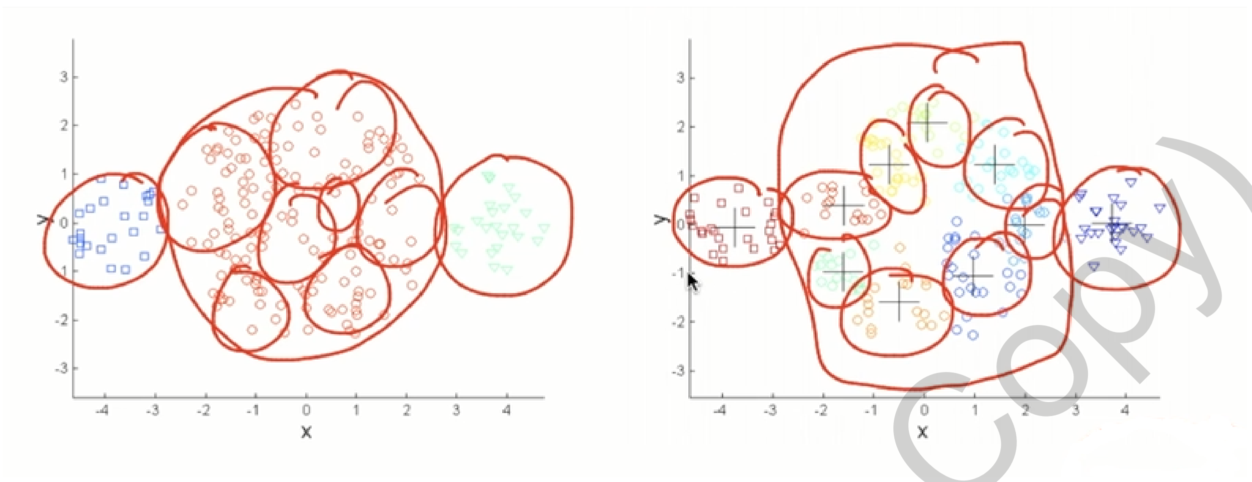
Whenever we have the points in non globular shapes, then the K-means algorithm mostly creates clusters in a convex/globular shape, and most of the times such clustering goes wrong.

2) K-Means has a problem when our data has outliers

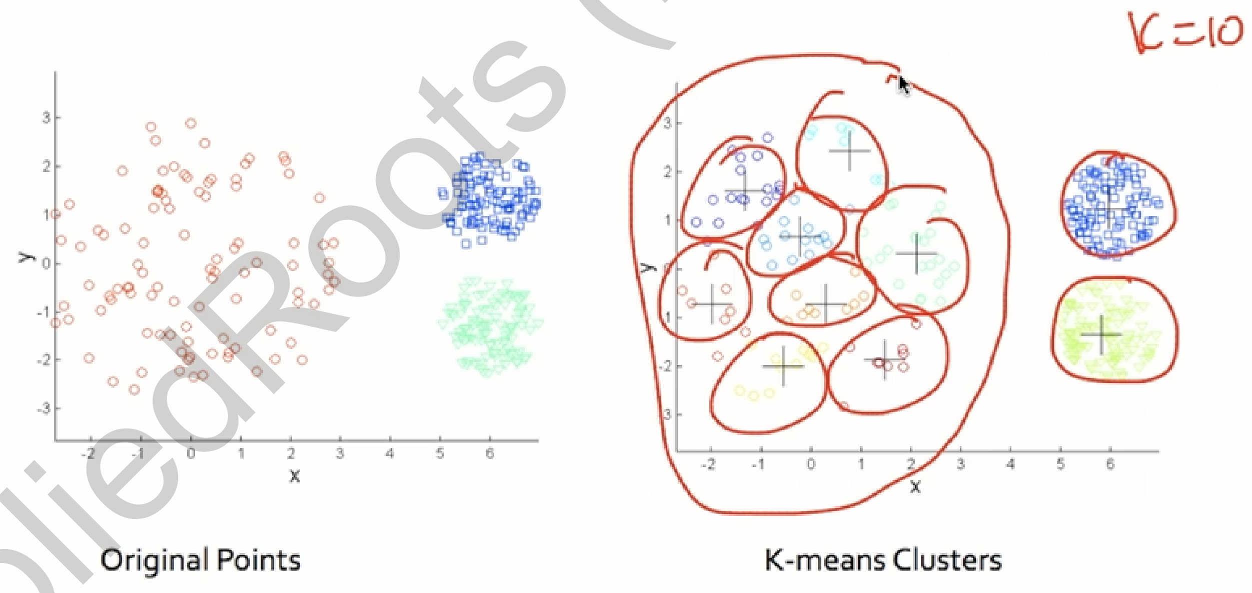
Overcoming the limitations of K-Means

1) When we are given the points with different sized groups, one solution is to increase the number of clusters. After more clusters are formed, we need to combine the similar clusters which is not an easy task.

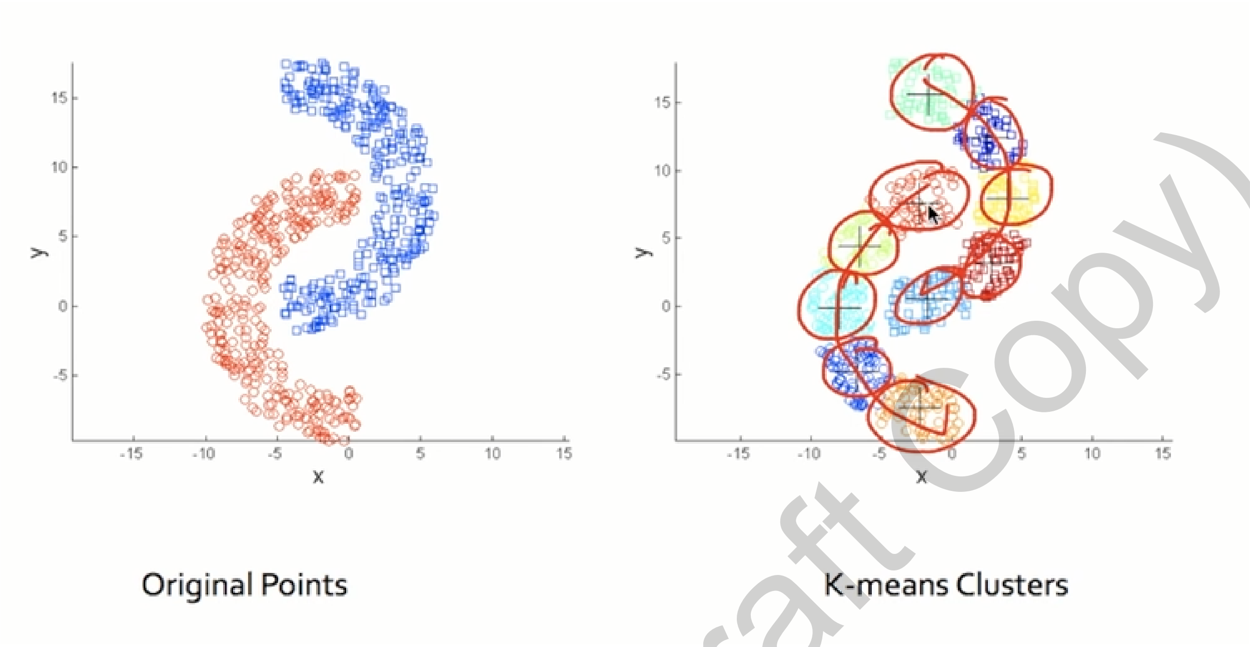
   Evaluating clustering is not an easy task as we do not have supervised output. We depend only on the intra-cluster and inter-cluster distances.



2) Whenever we have the data points with different densities, then if we increase the number of clusters, the smaller and denser clusters will remain the same, whereas the larger and sparse clusters get divided into multiple small sized clusters. It is again difficult to combine all these smaller clusters into a larger one.



3) When we are given the points in non-convex shape, then increasing the number of clusters would split the nearby into smaller clusters as shown below.



Increasing the number of clusters is always not the solution, but it makes sure that the points belonging to different clusters do not get combined together. If we want to use only K-means for our problem, then only we have to go with these steps. Otherwise, we can go for any other clustering technique.

50.10 - K-Medoids

One of the basic problems with the classical KMeans (or) KMeans++ is the centroids C_1, C_2, C_3, …., C_k that are obtained may not be interpretable.

For example, let us take Amazon Fine Food Reviews dataset into consideration. In this dataset, 'x_i' is represented by a review text converted into vector form.

The centroids C_1, C_2, C_3, …., C_k are computed using the mean points. In text to vector form notation, each of the coordinates represent the features and as the coordinates of these centroids are computed by taking the average of the corresponding coordinates of various points, these centroids are not interpretable.

Instead of giving the centroids C_j's that are computed using the mean, if a data point $x_j \in D$ is given as the first centroid, we can read this review and understand. The data point 'x_j' is more interpretable than 'C_j'.

If each centroid is a data point in the dataset 'D' (ie., $C_i = x_j \in D$), then the corresponding K-means is referred to as **K-Medoids**. In K-Medoids, we have the actual data points as the centroids, instead of the averages. It is a popular algorithm to interpret the centroids.

Partitioning Around Medoids (PAM)

1) Initialization

Same as KMeans++. Using the probabilistic method, we pick 'K' points from the dataset as Medoids.

2) Assignment

We use the closest medoid technique. (ie., $x_i \in S_j$ if medoid_j is the closest medoid to 'x_i')

3) Recompute/Update

In K-Medoids, for each medoid

a) Swap each medoid with a non medoid point.

b) If the loss decreases after the swap, keep the swap. Otherwise undo the swap.

The loss we want to minimize = $\boldsymbol{\Sigma_{i=1}^{k} \Sigma_{x \in Si} ||x\text{-}m_j||^2}$

Where m_j → Medoid 'j'

For example, if we are working on K(=2) medoids,

a) If $m_1 = x_1$, and $m_2 = x_6$, then loss = L_1

b) After swap, $m_1 = x_2$, and $m_2 = x_6$. Compute the loss value. Let it be 'L_2'.

If $L_2<L_1$, then keep $m_1 = x_2$, and $m_2 = x_6$. Else $m_1 = x_1$ and $m_2 = x_6$. Keep doing this for all the other pairs. If there are 'n' data points and 'K' medoids, we have to perform this nC_k times. Once the swap is successful, we have to go back to the assignment stage and fix these new values as the medoids.

But we know the medoids are also the data points from the dataset. So $||x_i\text{-}m_j||^2$ is the distance square between 'x_i' and 'm_j'. So if the distances between the points are given, then it becomes Kernel matrix. We also can minimize the loss using the kernel matrix.

So the K-medoids approach makes the centroids more interpretable and also allows kernelization. These are the advantages of using the K-medoids approach.

The centroids obtained by computing the averages are not interpretable because the mean of the original vectors may not make any sense as we may have non-integer values in some cells and we cannot construct the actual central point in each cluster.

Medoid based approaches ensure that the central point is always one amongst the original data points in the dataset and are easily interpretable.

In the case of KMeans, in which the centroid is computed by taking the mean of all the points in the cluster, which could make it completely uninterpretable. In K-medoids, we do not create any artificial points. The PAM approach is computationally expensive and would not work with the large datasets.

50.11 - Determining the right ‘K’

In K-means, ‘K’ is a hyperparameter and it can be determined through

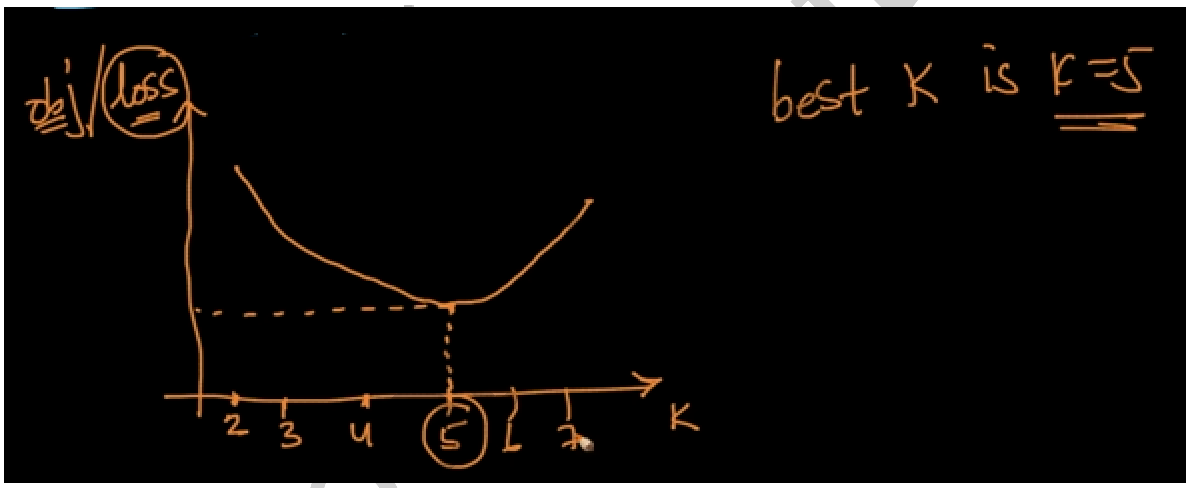
a) Domain Knowledge

If our problem contains a fixed number of clusters/groups all the time, then ‘K’ will be determined using the domain knowledge.

b) Elbow Method (or) Knee Method

The objective function of K-Means clustering is to minimize the loss and it is given as

$\boldsymbol{\Sigma_{i=1}^{k} \Sigma_{x \in S_i} ||x-C_i||^2}$



The value of ‘K’ that yields the least value for loss is considered to be the optimal hyperparameter.

Note

As the video lecture 5.12 is all about code sample discussion, we are not providing any notes for it. You can go through the lecture, and for any queries, please feel free to post them in the comments section under the video lecture (or) you can mail us at mentors.diploma@appliedroots.com.

50.13 - Time and Space Complexity

For K-means clustering, there is no train and test phase. It is a one time process. K-means is faster and simple.

Total Time Complexity = O(n*k*d*i) where
n → Number of data points
k → Number of clusters
d → Number of dimensions
i → number of iterations
The typical values are k<=10 and i<=300, and in such cases, the total time complexity is almost equal to **O(n*d)**.

Total Space Complexity = O(nd+kd), here
nd → Space required to store the data points
kd → Space required to store the centroids
When k<=10, the total space complexity is almost equal to **O(nd)**.