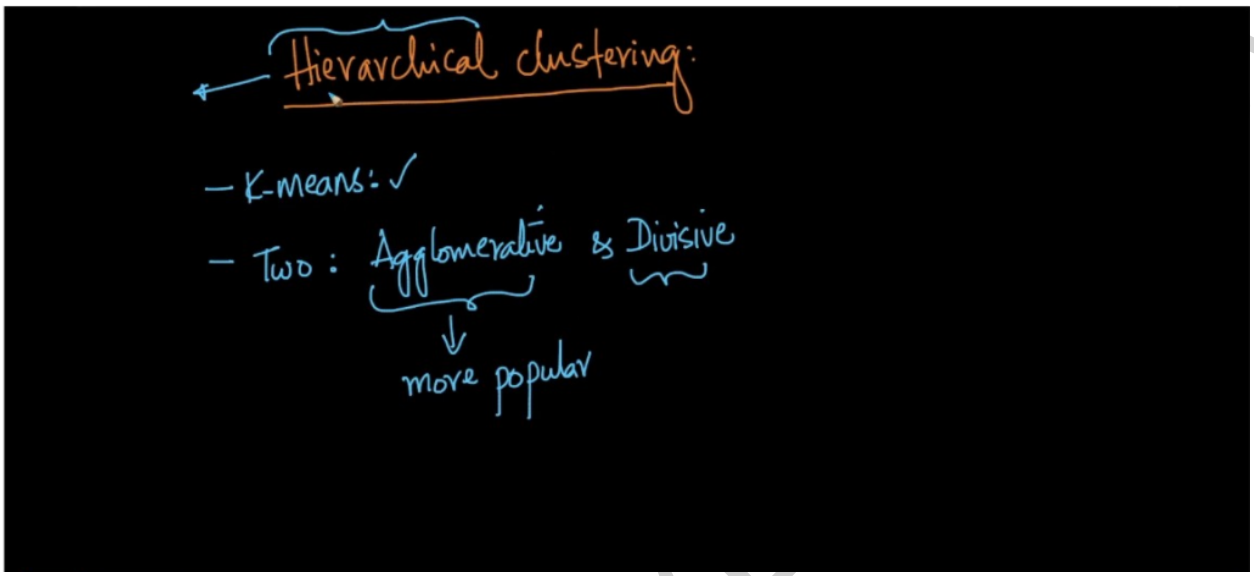
51.1 Hierarchical Clustering

In this chapter we will look at an algorithm called **Hierarchical Clustering.**



Timestamp 0:42

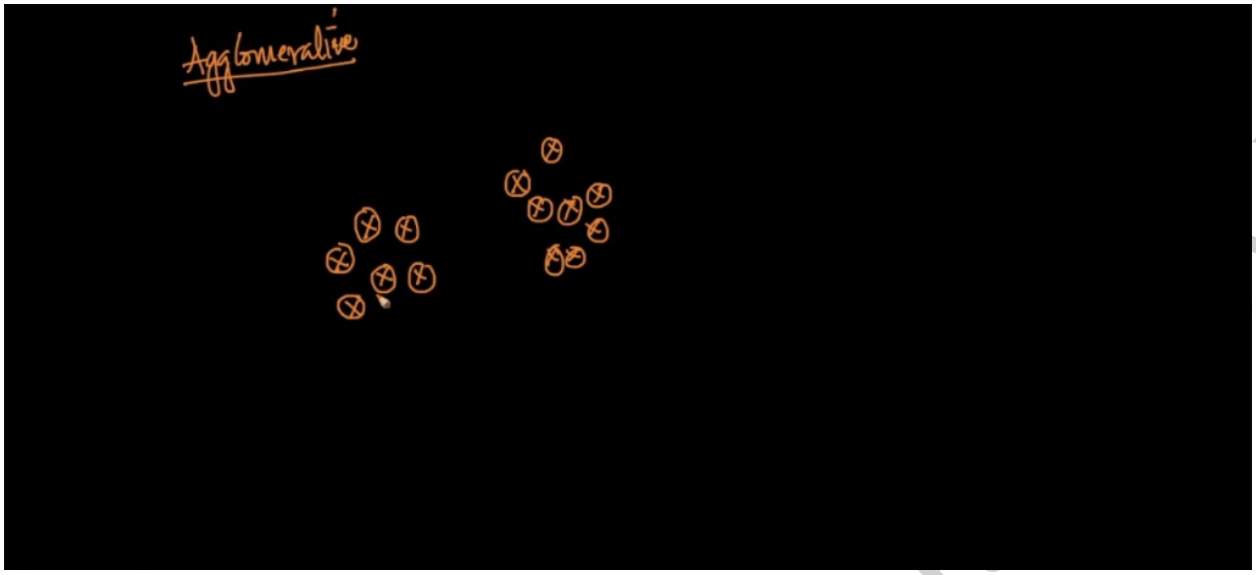
We already learnt about the **k-means clustering algorithm** in the past.

There are two types of **hierarchical clustering** :

1.) **Agglomerative clustering** -> The more popular one.
2.) **Divisive**

Example for **agglomerative clustering** :

Let’s consider a simple example. Let’s assume we have a bunch of points as shown below.

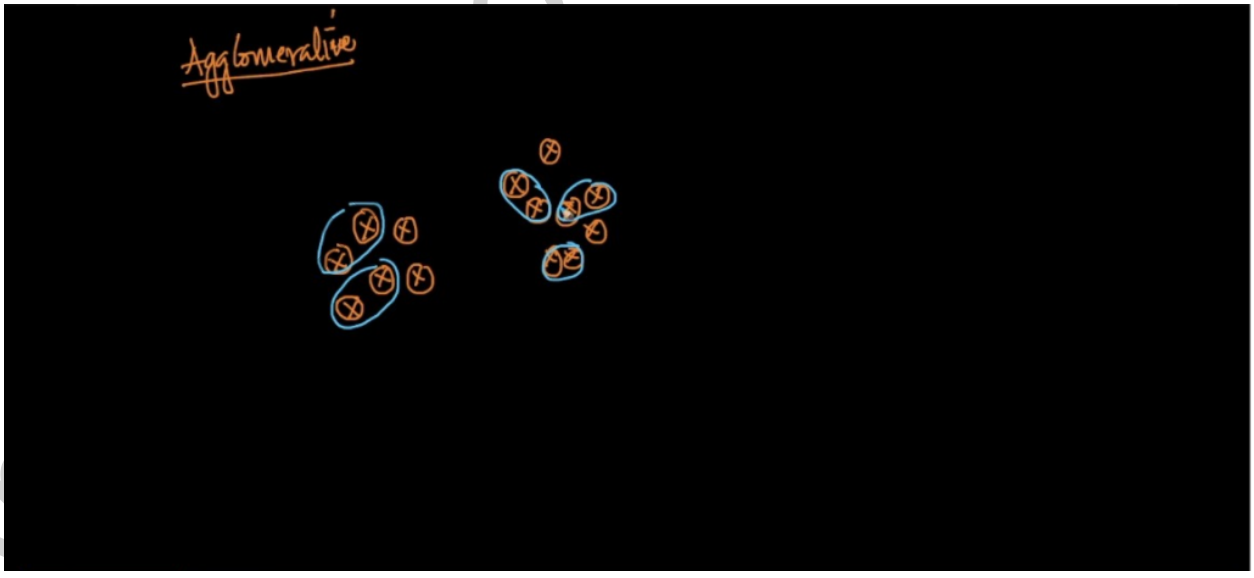


Timestamp 01:28

1.) At first, each point on it's own is a cluster.

2.) Then, for each point, we find it's closest neighbour (in terms of euclidean distance) and group it.

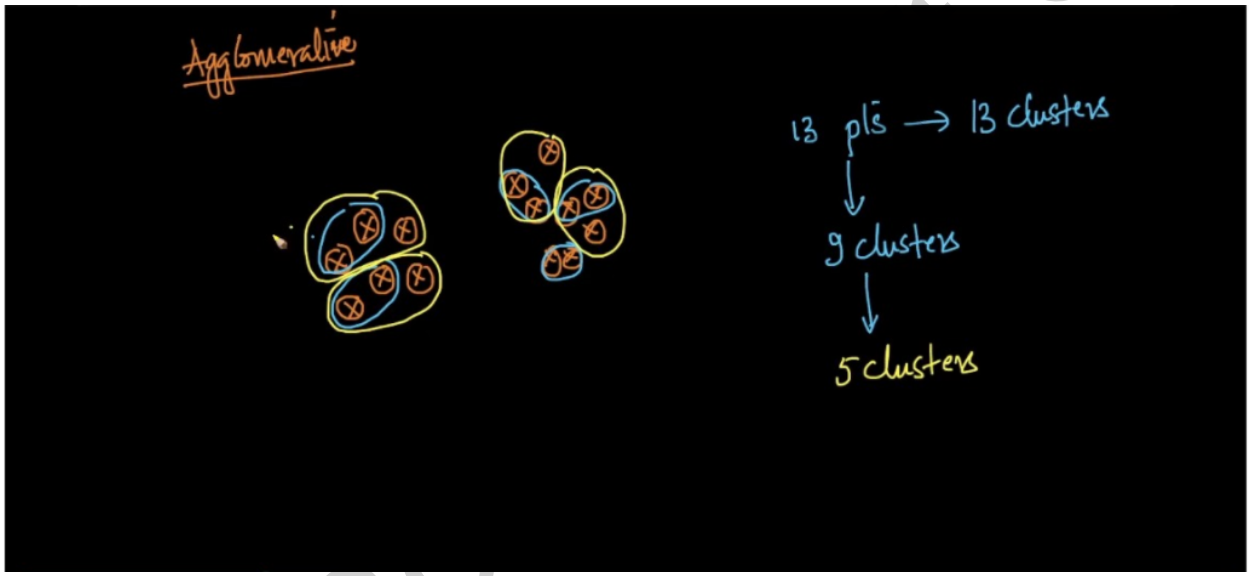
After the first grouping is done.



Timestamp : 01:48
Note: Some points are not grouped together.
A cluster can also have one point. So, we made 9 clusters from 13 points.

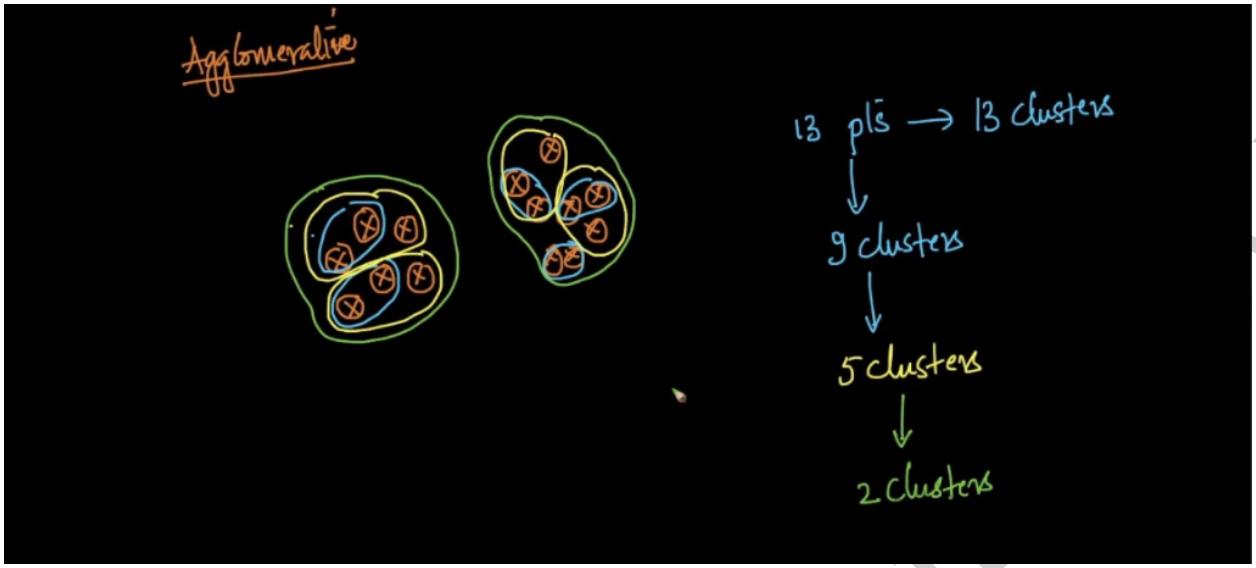
Again, we can perform the grouping by finding the neighbours. Now, we are going to find the distance between two clusters rather than two points before.
How to find it ?
Ans : We can take the mean of all points present in a cluster c . Then we can represent a cluster by this mean point. Now, do this for all clusters. So, we get a mean point for each cluster. Again, repeat the steps mentioned previously.

After the second grouping is done.



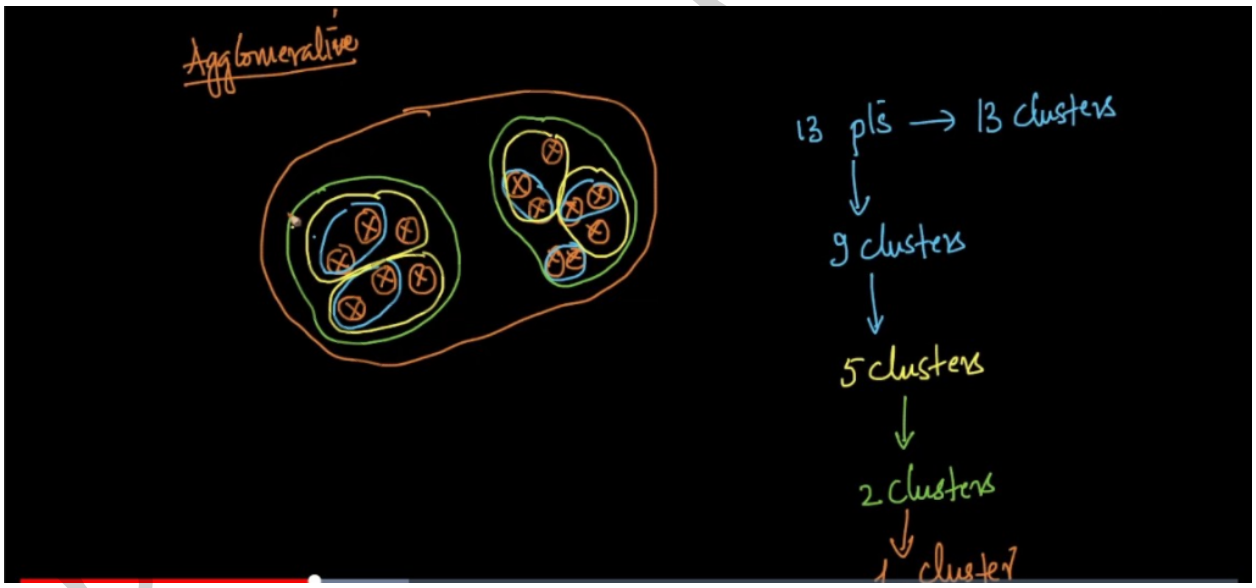
Timestamp : 02:41
Now, we made 5 clusters from 9 clusters before. Again, we can incrementally group the clusters by following the previously mentioned step(s).

After the third grouping is done.



Timestamp : 03:00
Now, we got 2 clusters from 5 clusters before. At the last stage, we simply group the two clusters together.

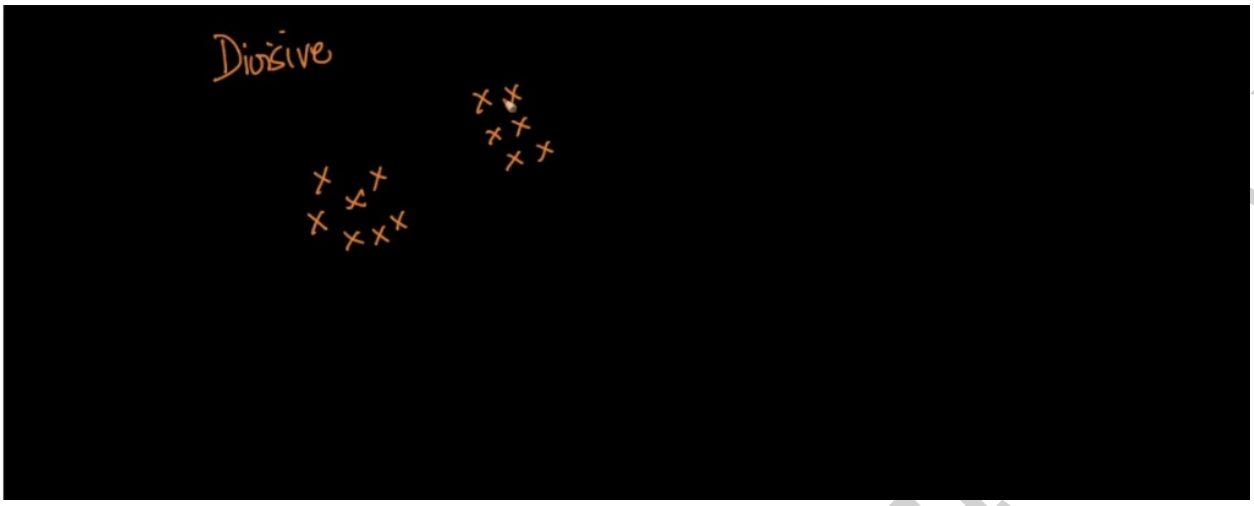
After the fourth grouping is done.



Timestamp : 03:12
Finally, we got 1 cluster from 2 clusters before. After this, we can't group anymore. In this stage, the algorithm stops.

Example for **divisive clustering** :

Let’s assume we have a bunch of points as shown below.

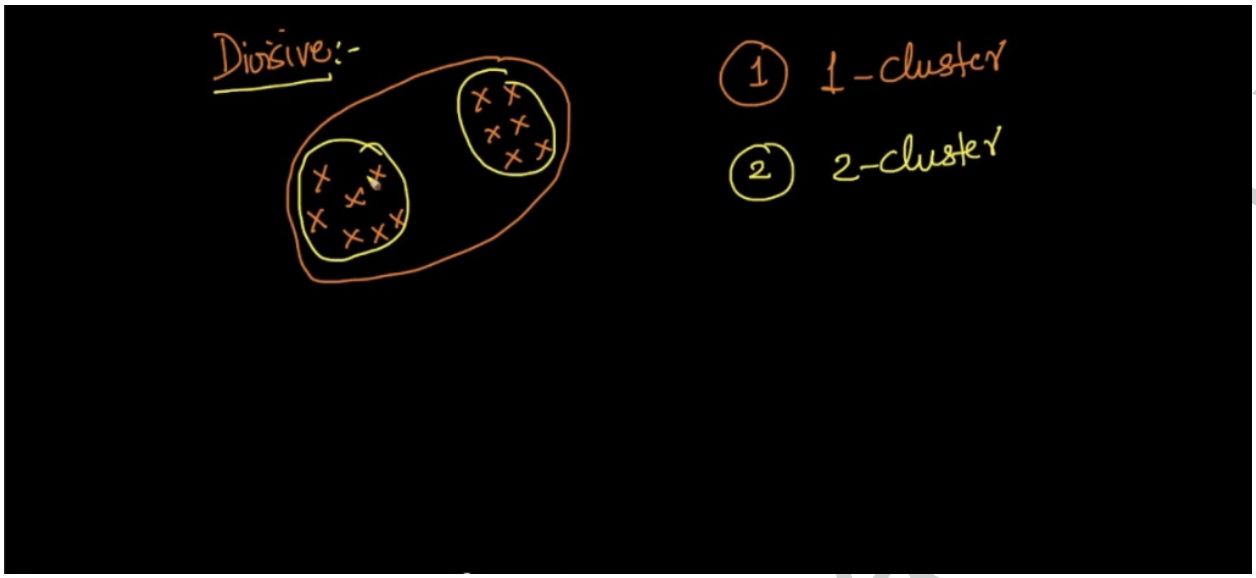


Timestamp: 04:25

1.) At the first iteration, the divisive algorithm considers the entire points as one cluster as opposed to agglomerative , where each point as it’s entire own is a cluster.

Timestamp : 04:50

2.) In the second iteration, we divide the cluster into two parts.



Timestamp : 04:53

3.) From the second iteration, we have two clusters. In the third iteration, we will divide each of those two clusters into two clusters. So, we get 4 clusters altogether.

Timestamp : 05:10

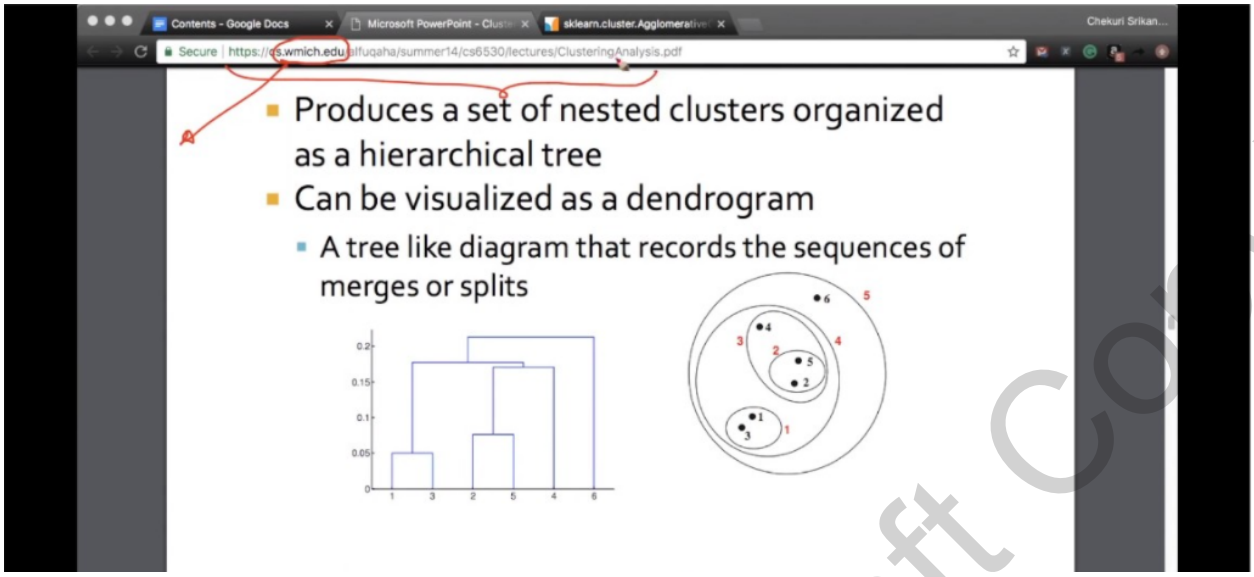
4.) Now, we can repeatedly perform the division operation. At the end, we will have 13 clusters altogether.

Timestamp : 05:21

This algorithm is exactly opposite to the agglomerative algorithm which we have studied earlier. But, agglomerative algorithms are the most popular one.

- In an agglomerative algorithm, we have some criteria to group together points to form a cluster. One such criteria is the euclidean distance between two points. There are also other criteria like cosine distance, manhattan distance ,etc.
- In a divisive algorithm, we also need some criteria to decide how to divide the clusters. Some methods are Ward's criterion, SSE , etc.
- The key point in agglomerative clustering is the choice of similarity measure or distance measure. This really affects the performance of the clustering.

We can also visualize the process of agglomerative clustering in the form of a dendrogram. Please refer below.

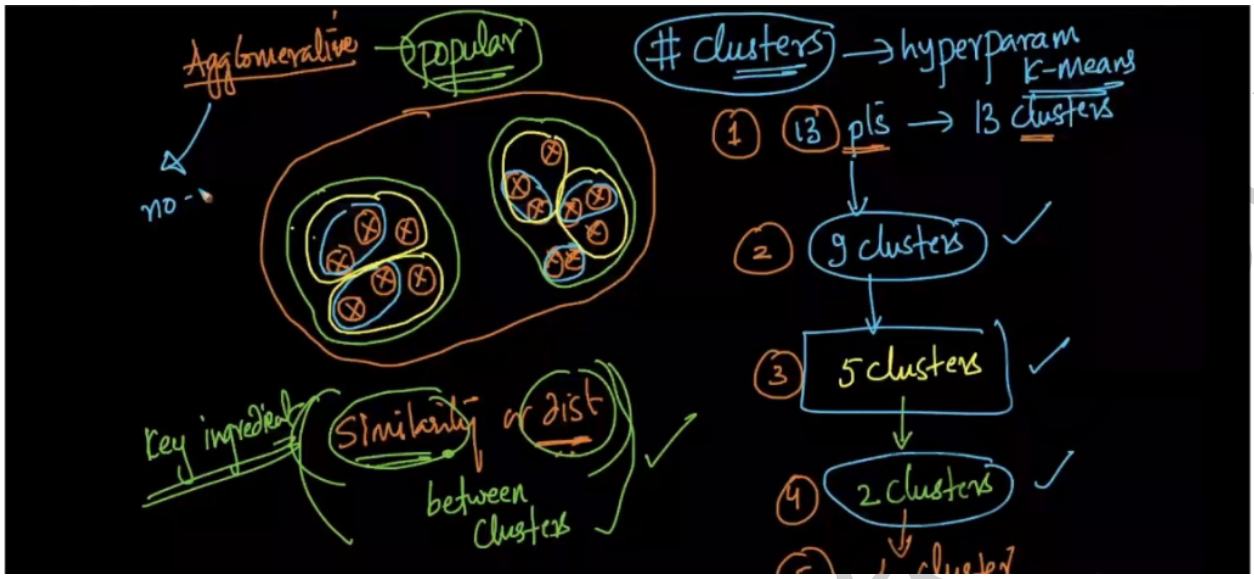


Timestamp : 07:21

1.) At first, point 1 and point 3 are grouped together into a single cluster.
2.) In the second iteration, point 2 and point 5 are grouped together into a single cluster.
3.) In the third iteration, the point 4 and the cluster formed in the second iteration are grouped together into a single cluster.
4.) In the fourth iteration, the cluster formed in the third iteration and the cluster formed in the first iteration are grouped together into a single cluster.
5.) In the final iteration, the cluster formed in the fourth iteration and the point 6 is grouped together into a single cluster.

Important points :

1.) In agglomerative clustering, the number of clusters is not a hyper parameter as opposed to the k-means clustering algorithm.
2.) We can stop the process at any point and simply return the number of clusters made till that point.

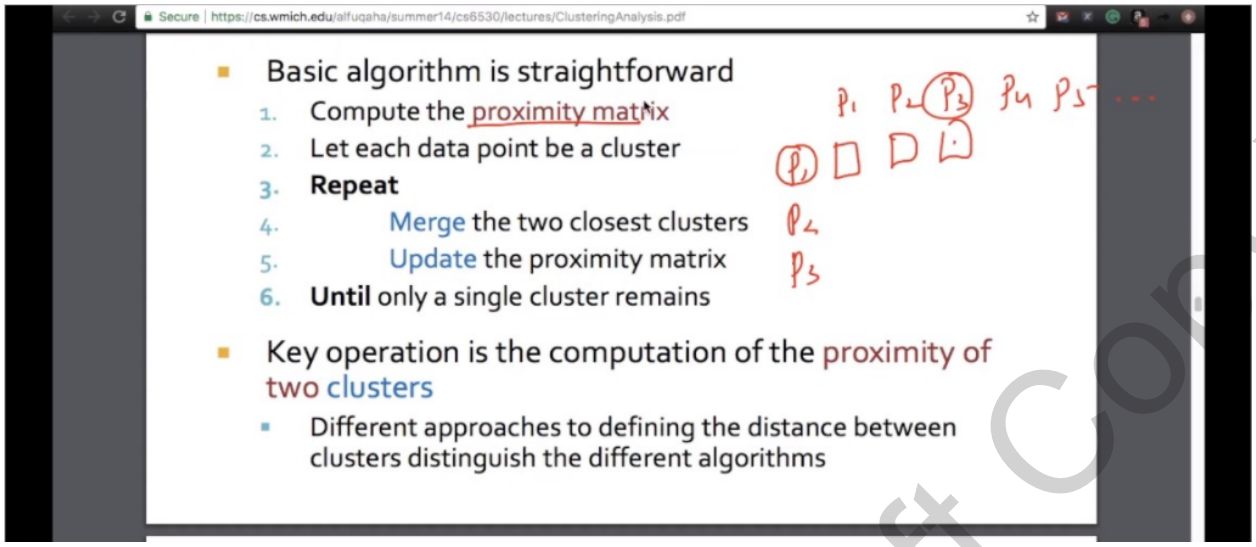


Timestamp : 11:49

We can stop at any number of clusters and return. For example, if we want 5 clusters to be made from the points, then we can stop at the third iteration and simply return those 5 clusters.

Suppose, if we trained the k-means clustering algorithm for 5 clusters. If we changed the mind to 4 clusters, then we need to train the k-means from scratch for the 4 clusters. But in agglomerative, it's not so.

51.2 Agglomerative Clustering



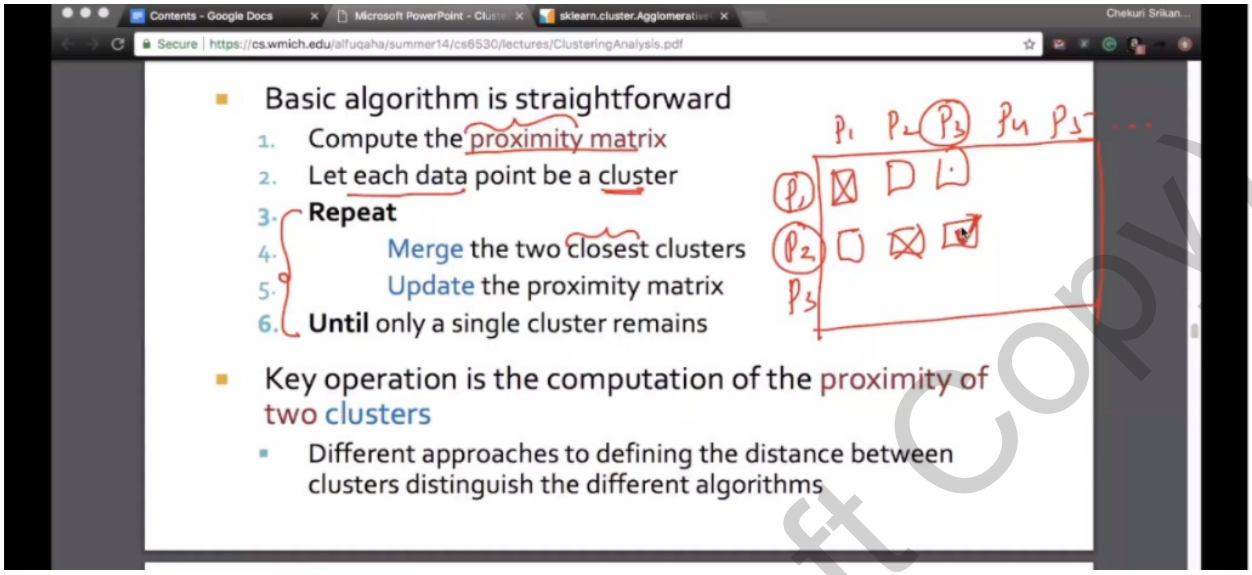
Timestamp : 0:41

1.) First, we need to compute the proximity matrix. Assume we have n points p1,p2,p3..pn. We will define a n*n matrix P where each entry Pi,j describes the proximity between point pi and point pj. For example, P1,2 denotes the euclidean distance between point p1 and point p2. As described above in the image, we will have a matrix P of size n*n. The entries in the diagonal are always zero. This is because , distance between a point pi and itself is zero.

2.) At first, each data point is treated as a cluster.

3.) Repeat:

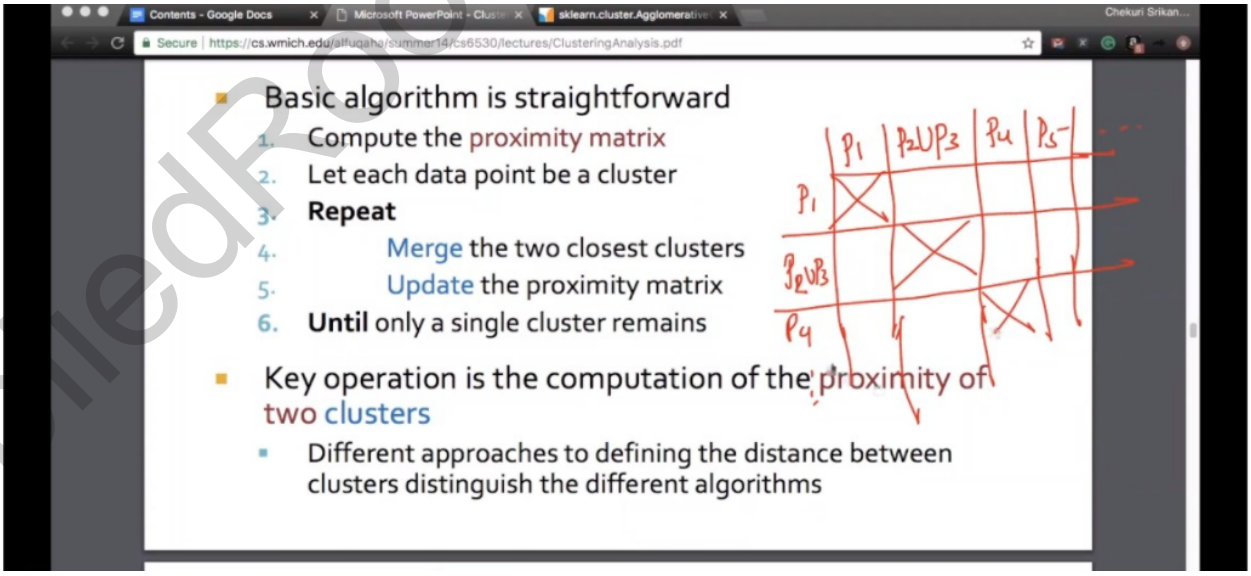
1.) Merge the two closest clusters -> At first, find the smallest value in the P matrix. Let's say that the smallest value is present in the Pi,j entry. This means, the point pi and point pj are close among all other pairs. So, we group the two points into a single cluster. Now, simply mark the entry Pi,j as done i.e, it's a cluster.



Timestamp : 02:00

As we can see from the above image, points p2 and p3 are grouped into a single cluster.

2.) Update the proximity matrix -> Now, we will update the proximity matrix since points p2 and p3 are grouped into a single cluster.



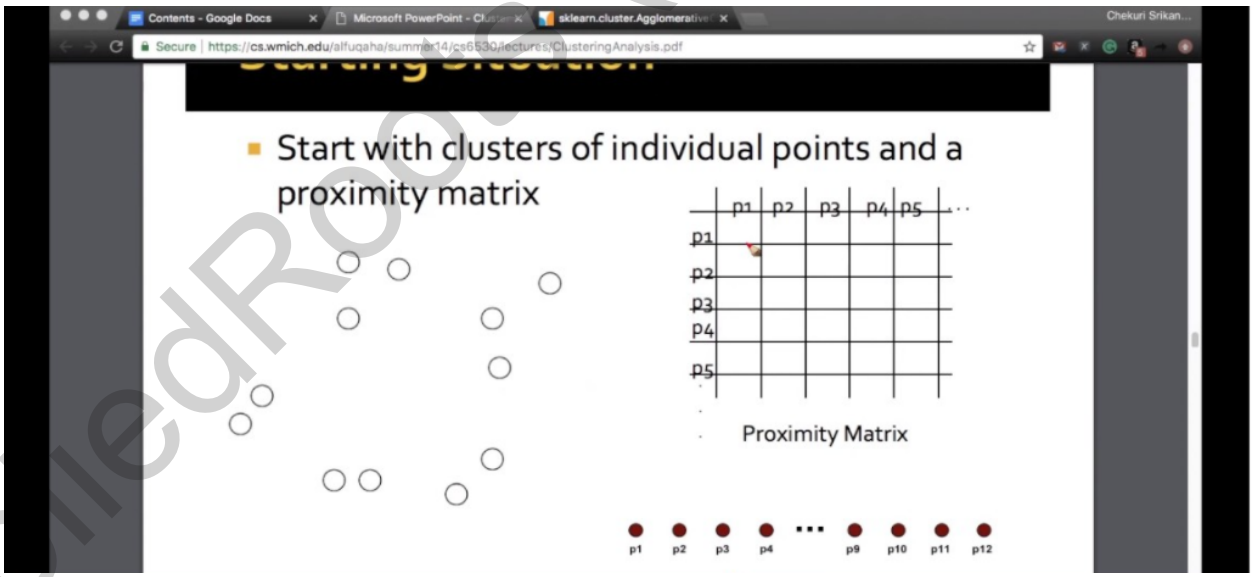
Timestamp : 02:48

As we can see from the above image, we have made P2UP3 as a column and row. This represents the fact that p2 and p3 belong to a

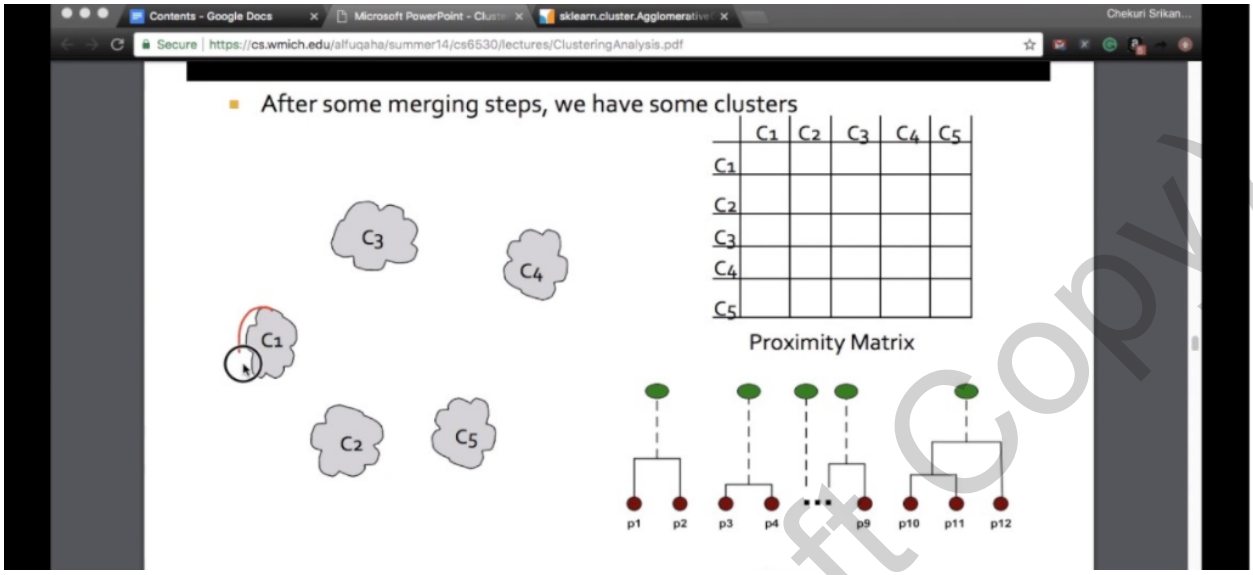
single cluster. Since we have added a new row and column, we need to also update the entries. We will update the entries where P2UP3 is present and will not update the others since that is not affected by the change. For example, we don't need to update P1,4 as the distance remains unchanged. But we need to update Pp2Up3,2 as it's now the distance between the cluster p2Up3 and the point p2.

So, now we need to find a method to compute the distance between the cluster and a point p. For example, let's say we want to update Pp2up3,4 i.e, we need to find the distance between the cluster p2up3 and the point p4. We will first find the mean of the points p2 and p3. Let's call it p_mean_2_3. Now, we will find the distance between p_mean_2_3 and p4. This distance will be updated in the Pp2up3,4 entry. Basically, the mean point represents a cluster.

Let's see some illustrations describing the process.

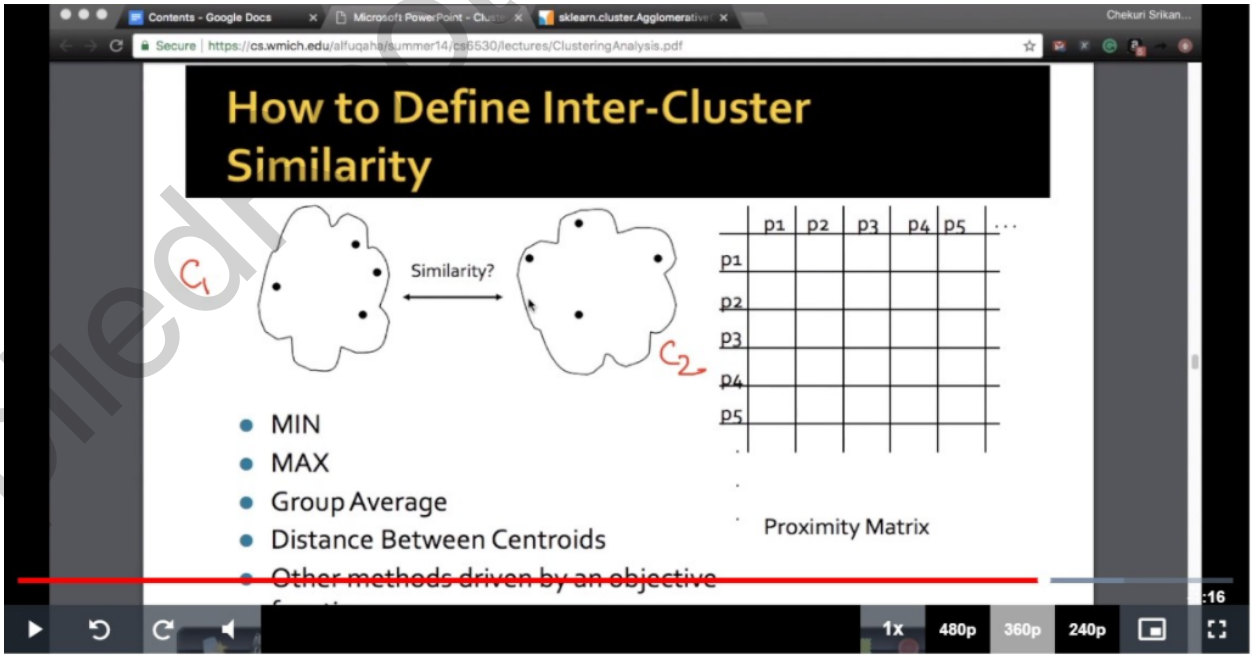


Timestamp : 04:50



Timestamp : 04:58

We already talked about one way of computing the similarity or distance between a cluster and a cluster or a cluster and a point. We did by averaging the points in the cluster and then we represented the entire cluster by the mean point. But, there are also other ways of doing it.



Timestamp : 06:51

The methods are :

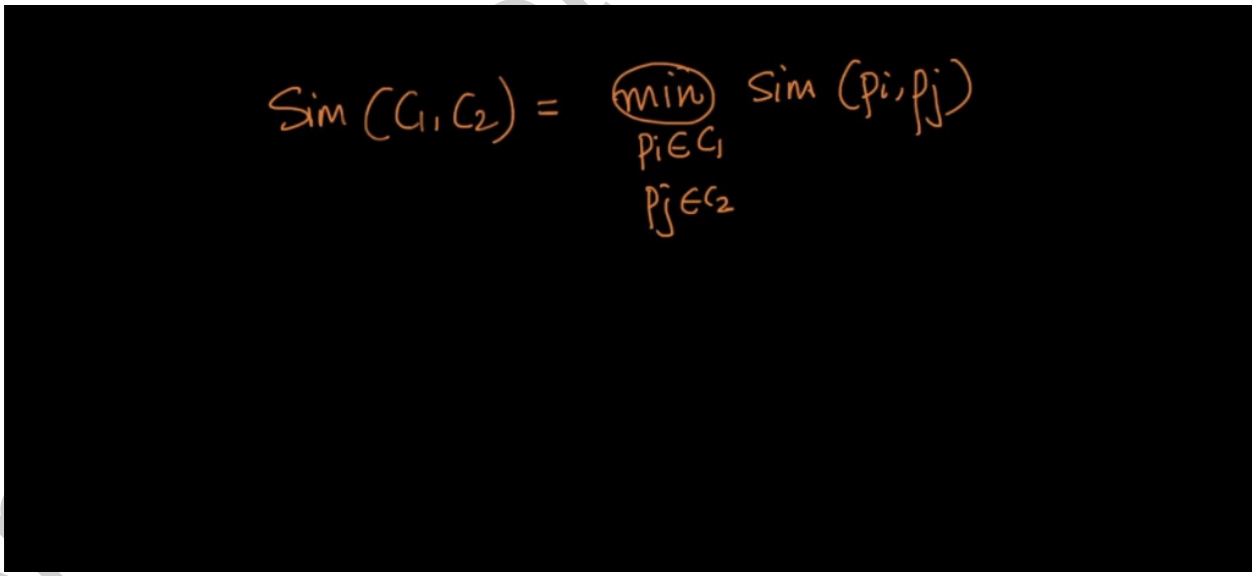
1.) MIN
2.) MAX
3.) Group Average
4.) Distance between Centroids
5.) Other methods driven by an objective

If you have time, please try to write the algorithm from scratch.

51.3 Proximity methods : Advantages and Limitations

In this lecture, we will see about How to define inter-cluster similarity between two clusters.

1.) MIN :



Timestamp : 01:51

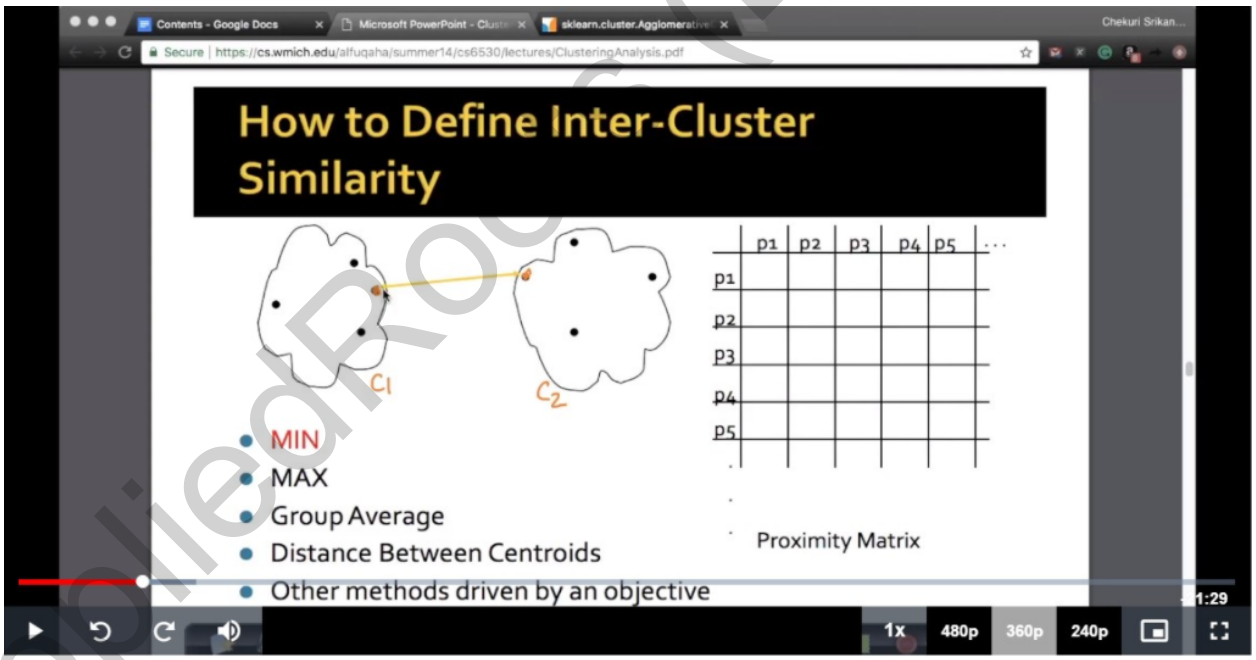
Similarity (C1,C2) where C1 and C2 are any clusters is defined as :

Min Similarity (pi,pj) where pi belongs to C1 and pj belongs to C2. Here, similarity is nothing but the euclidean distance or any other valid metrics. For example cosine similarity, manhattan distance, etc.

In simple terms, create a set C1 where the set has all points belonging to C1. Also, create a set C2 where the set contains all the points belonging to C2. Now, do this:

Min_sim = infinity or a largest possible integer value in python.
For pi in C1:
    For pj in C2:
        Min_sim = min(Min_sim,Similarity(pi,pj) #Similarity(pi,pj) can be retrieved from the similarity matrix we discussed earlier.

Return Min_sim.



Timestamp : 02:26

In the above image, we have highlighted the points pi and pj which have the minimum similarity. It's clearly evident from the image.

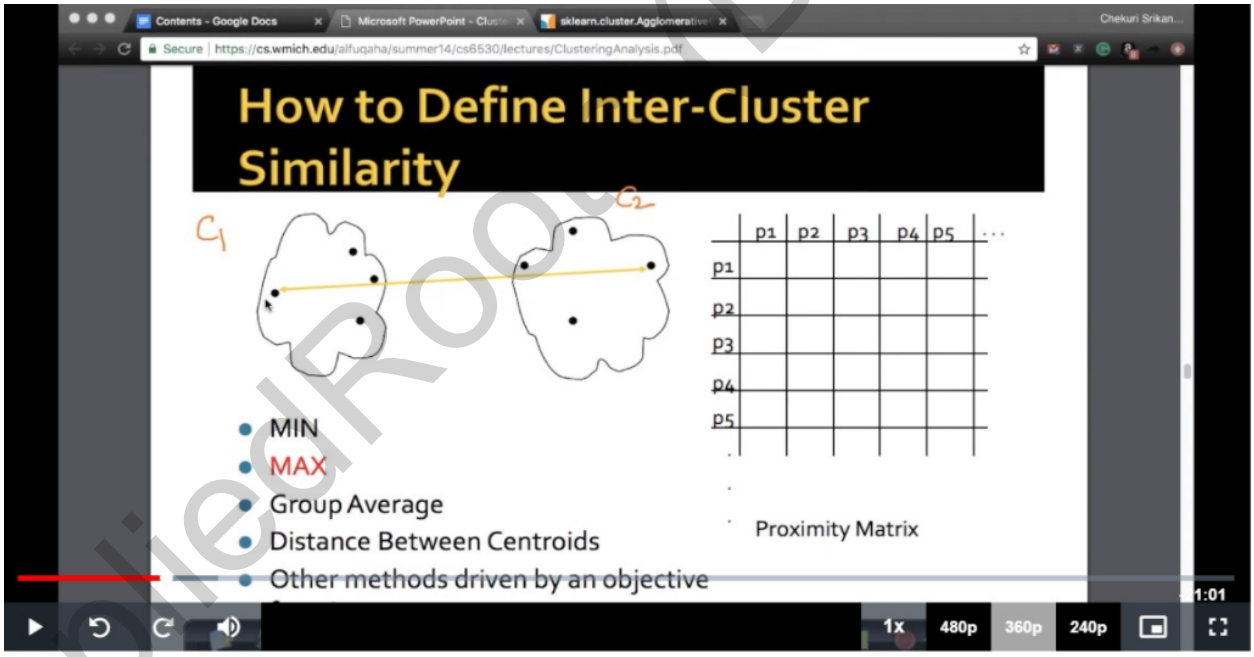
2.) MAX :

It's exactly the same as MIN. The only difference is that we are going to replace the min operation by max.

Please refer below for the procedure.

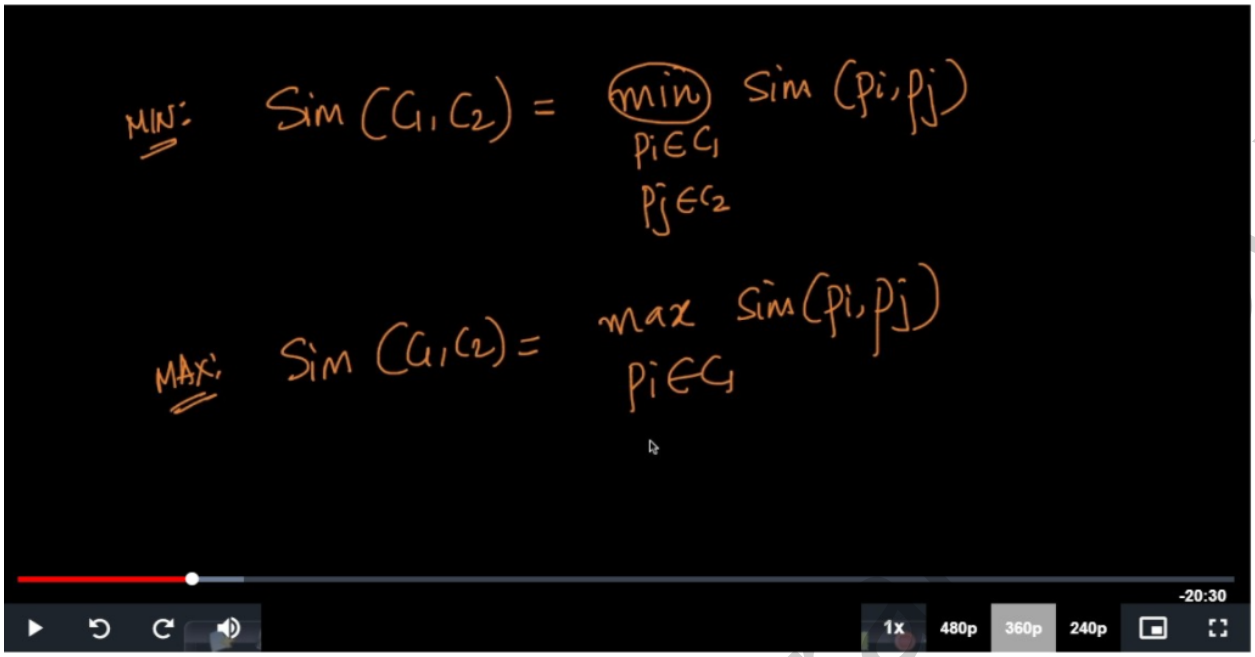
Max_sim = - infinity
For pi in C1:
    For pj in C2:
        Min_sim = max(Max_sim,Similarity(pi,pj) #Similarity(pi,pj) can be retrieved from the similarity matrix we discussed earlier.

Return Max_sim.



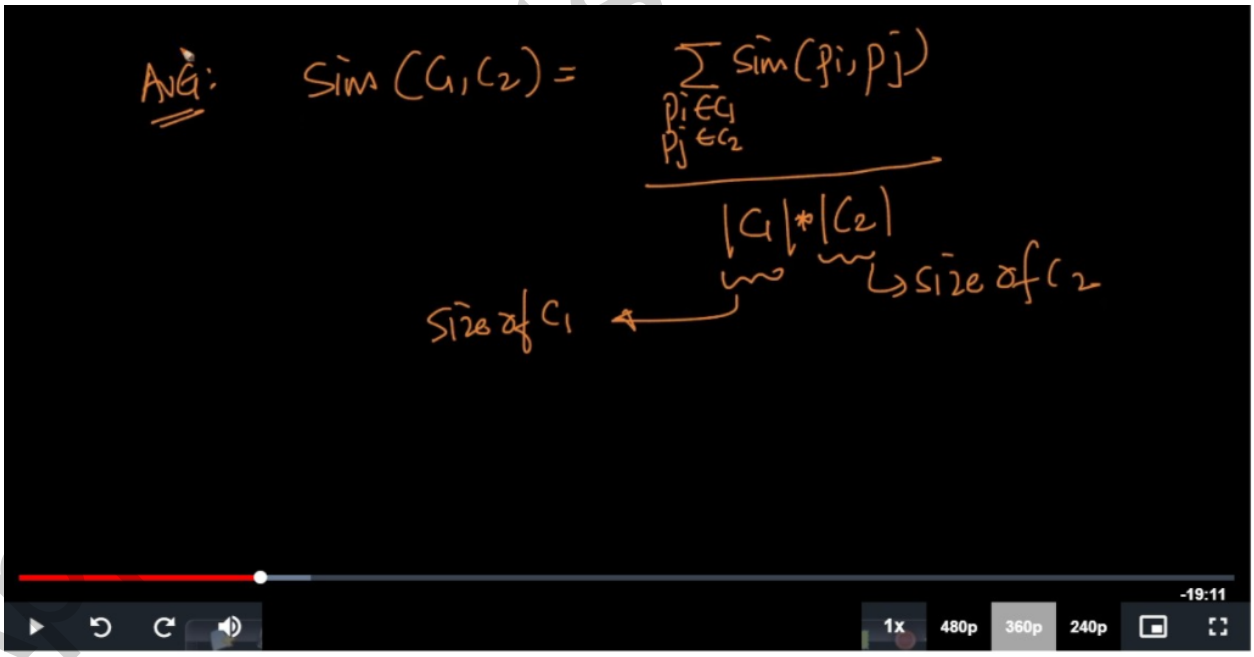
Timestamp : 02:54

In the above image, we have highlighted the points pi and pj which have the maximum similarity. It's clearly evident from the image.



Timestamp : 03:25

3.) AVG :



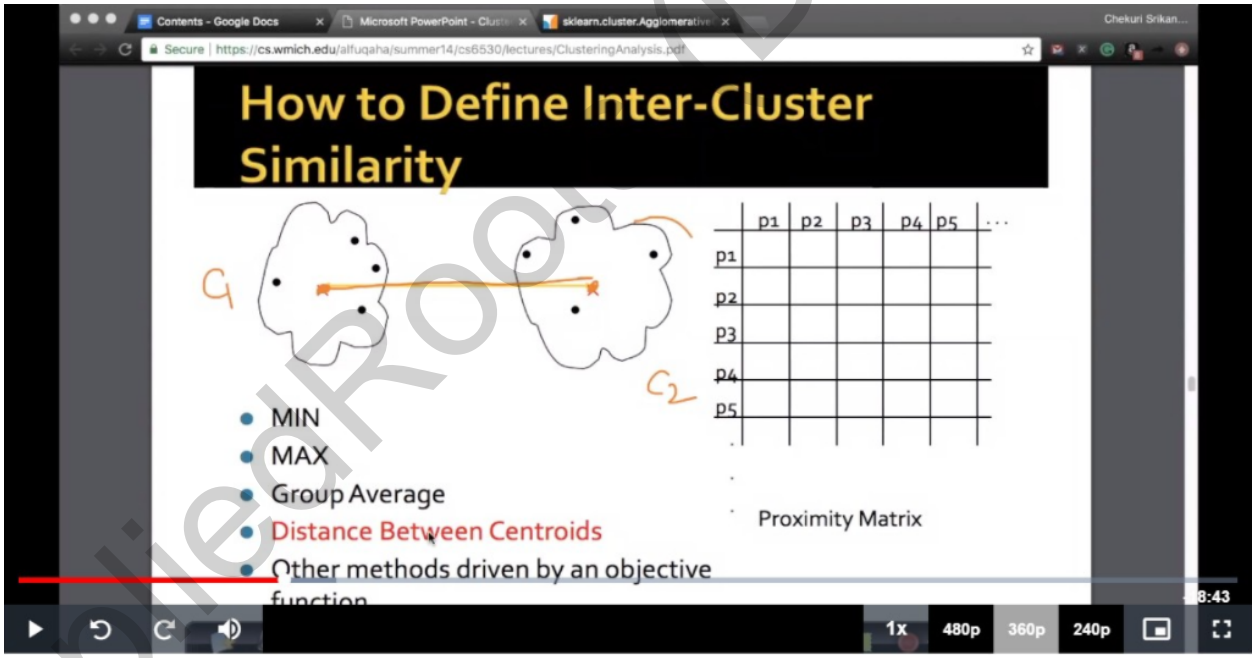
Timestamp : 04:44

Please refer below for the procedure

```
Count_of_points = 0
Sum_of_similarity = 0
For pi in C1:
    For pj in C2:
        Count+=1
        Sum_of_similarity += Similarity(pi,pj) #Similarity(pi,pj) can be retrieved from the similarity matrix we discussed earlier.

Return Sum_of_similarity/Count_of_points
```

In simple terms, we are adding all the similarity(pi,pj) for all i,j and then dividing by the total number of combinations if i,j we can make.

4.) Distance Between Centroids :



Timestamp : 05:12

This method is very simple and we discussed it earlier.

1.) First, create a set C1 where the C1 contains all points p which belongs to the cluster C1.

2.) Then, create a set C2 where the C2 contains all points which belong to the cluster C2.
3.) Now, simply take the average of all points in the set C1 and call it as p1.
4.) Now, simply take the average of all points in the set C2 and call it as p2.
5.) Now, compute the similarity between point p1 and p2 and return.

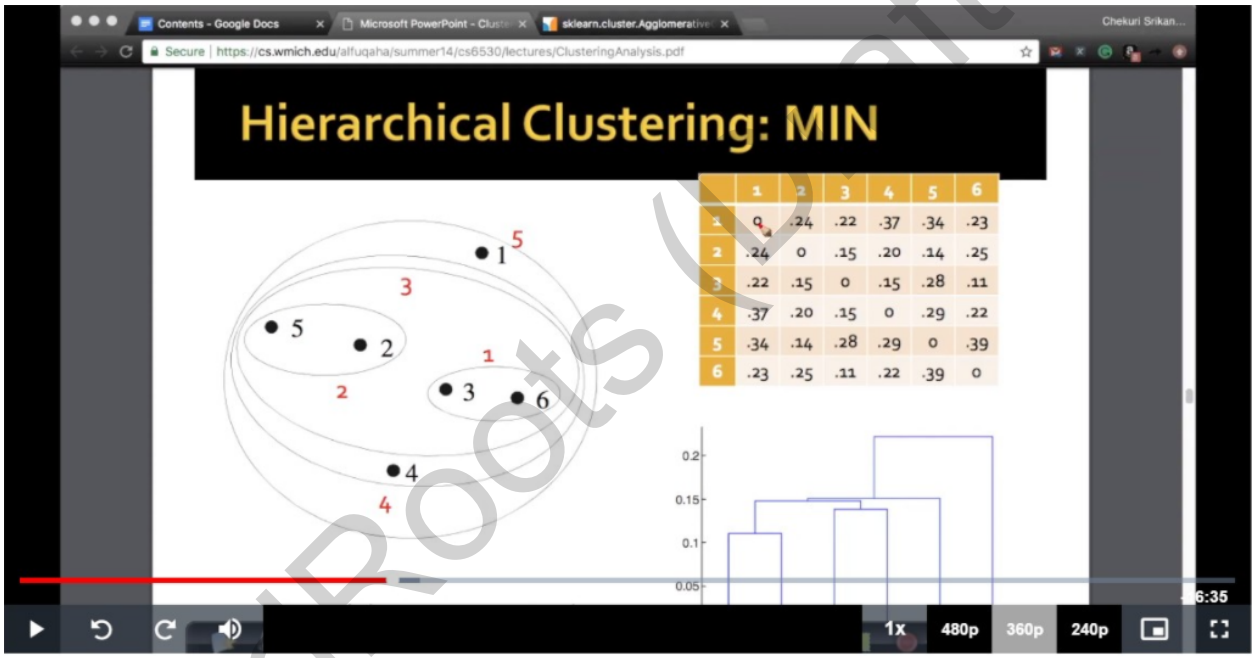
Algorithm :

```
mean_point_C1 = initialize this vector with zeros of size n
mean_point_C2 = initialize this vector with zeros of size n
count_C1 = 0
count_C2 = 0

For pi in C1:
    count_C1+=1
    For i = 1 to n:
        mean_point_C1[i]+=pi[i]
mean_point_C1/=count_C1

For pi in C2:
    count_C2+=1
    For i = 1 to n:
        mean_point_C2[i]+=pi[i]
mean_point_C2/=count_C2
```

The important thing in agglomerative clustering is the similarity. We need some function which can give us a similarity between points. One such function is the euclidean distance. The euclidean distance will be less for points which are close and be more for points which are far away. So, this gives some sort of similarity. Since , we are relying on similarity, we can kernalize it as we did in SVM.

We can’t apply kernelization in the method “Distance between Centroids”. Because , in that method, we are obtaining a new point (mean point). So, we will not have an entry in the similarity matrix or the kernel matrix if we are using kernelization. In theory, it’s possible to compute it. But , in practice , it’s hard. Because , everytime we obtain a mean point we have to compute the similarity for that. Because, in the similarity matrix, we will have the entries for the points which are present in the cluster. The mean point doesn't necessarily need to be present in the cluster.

Let’s consider an example for MIN .



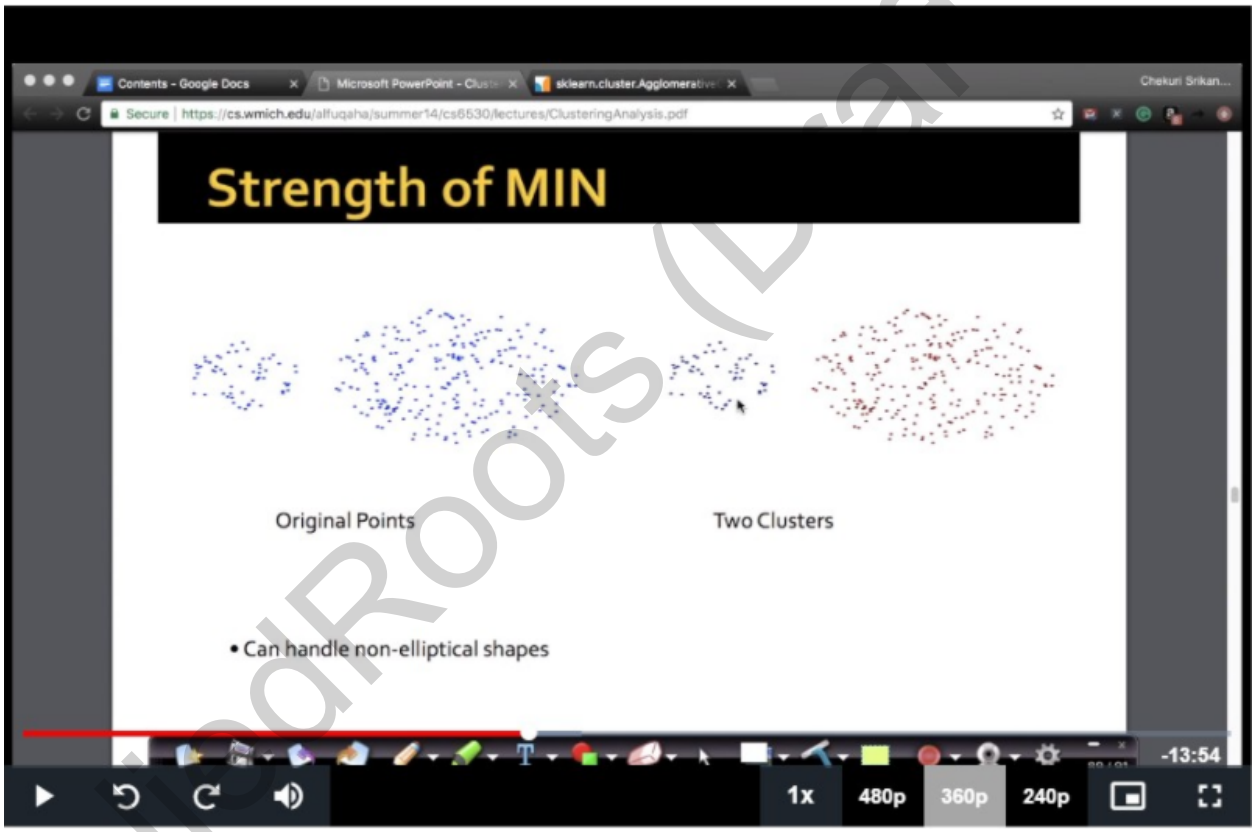
Timestamp : 07:20

We can see the similarity matrix defined above in the image. We can also see the dendrogram which we have studied earlier which records the sequence of operations we made while performing the agglomerative clustering.

1.) The diagonal entries are always zero. Since the distance between the point pi and itself is zero.

2.) First, we will find the minimum entry in the similarity matrix. We can see the entry 6,3 which has the minimum value of 0.11.
3.) So, we group points 3 and 6 into a single cluster.
4.) Again, we will perform the same steps as we discussed earlier.
5.) After performing the procedure, we will obtain the clusters as given in the above image.

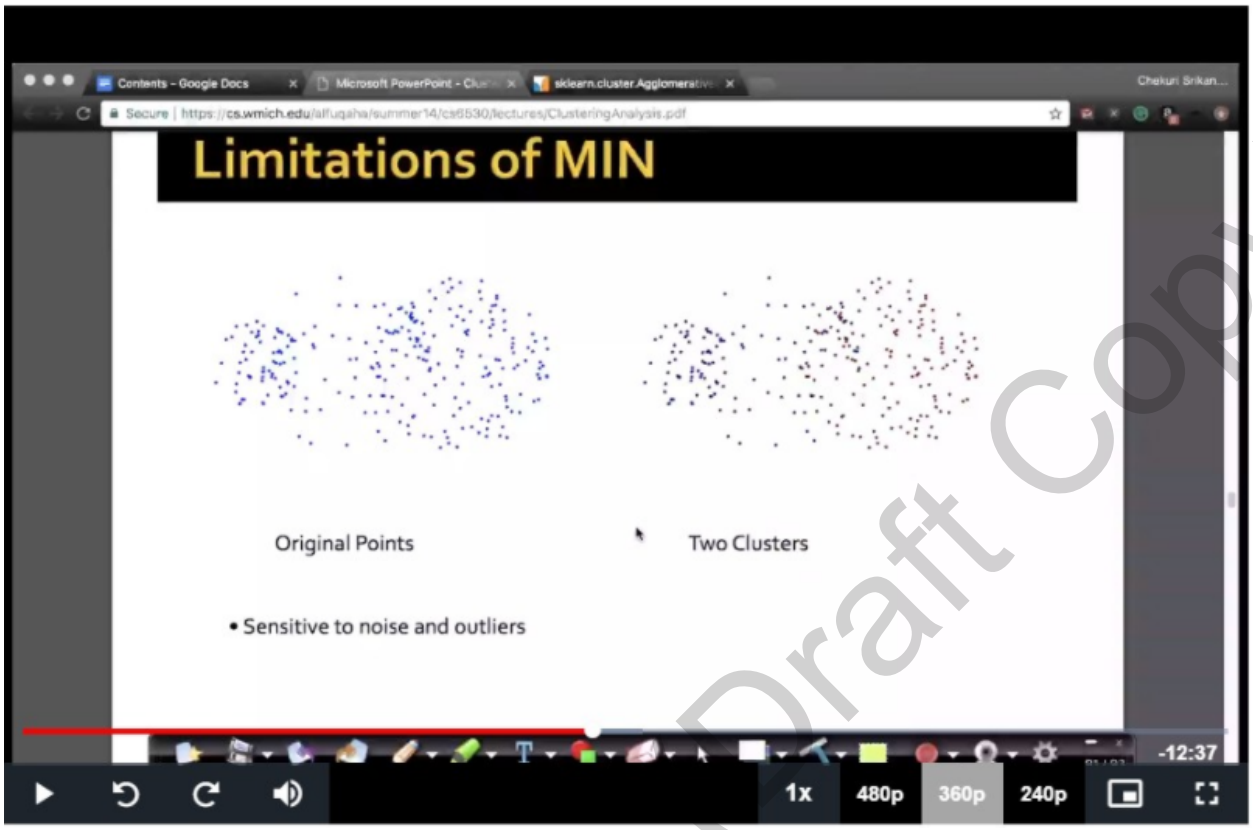
Strength of MIN clustering



Timestamp : 10:01

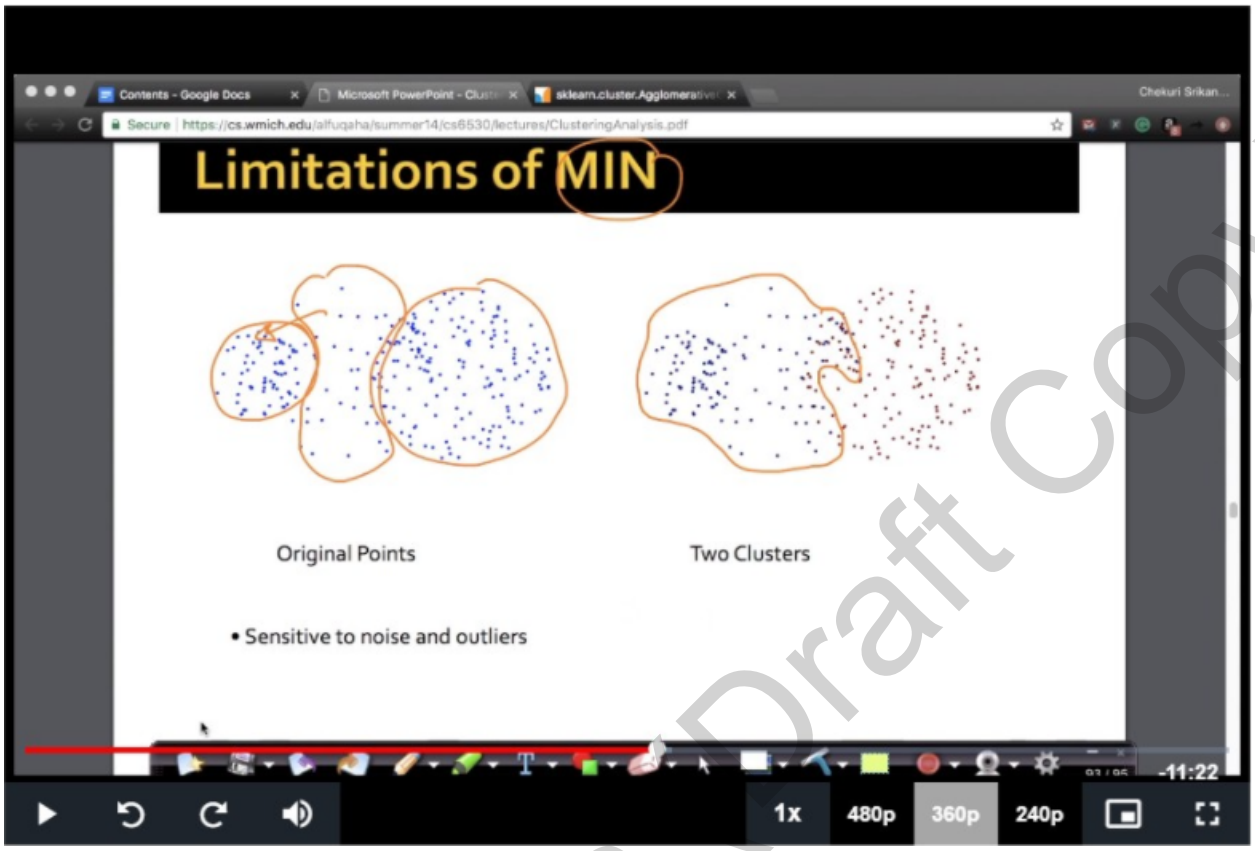
1.) It can handle non-elliptical shapes. As long as there's a separation between two clusters, it can handle very well.

Limitations of MIN



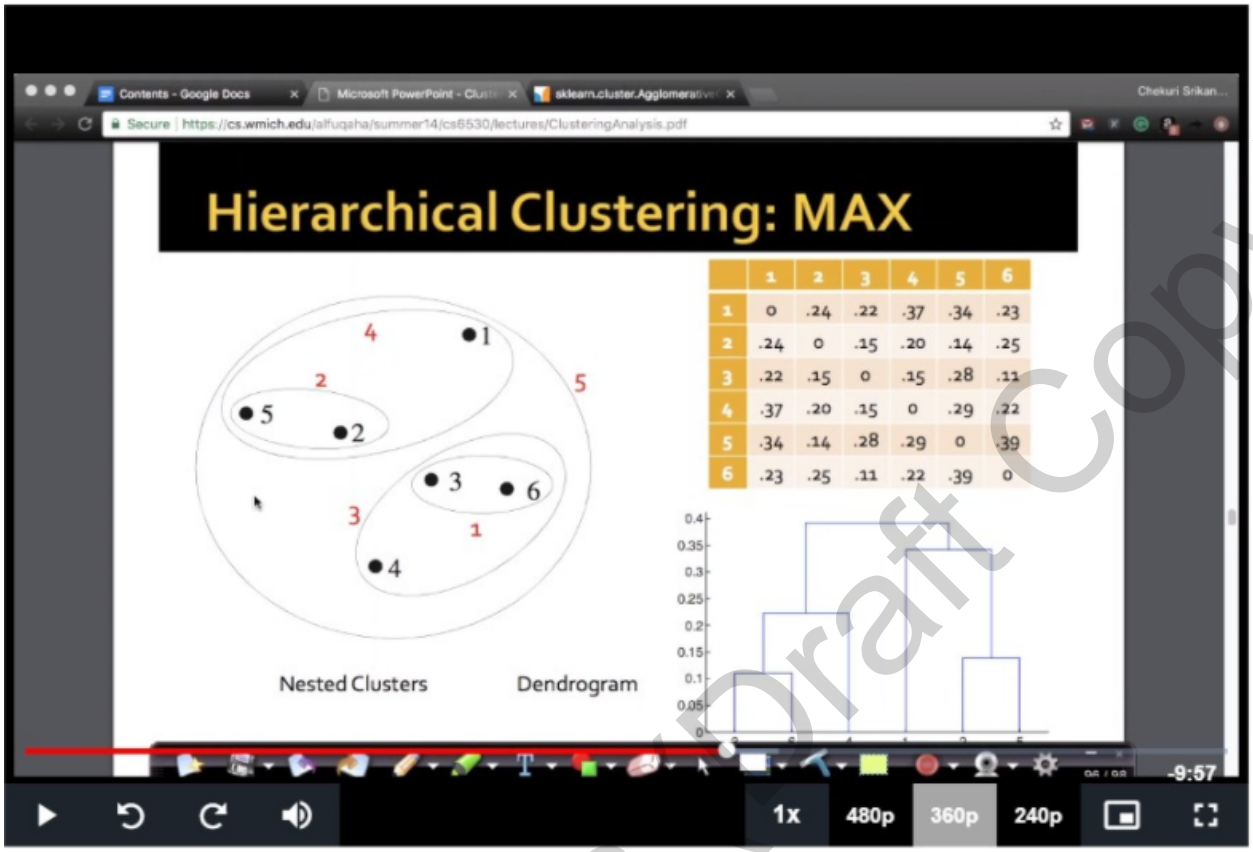
Timestamp: 11:18

1.) It's sensitive to noise and outliers.



Timestamp : 12:33
Since noise points are also closer to the clusters themselves, the algorithm will think that this point belongs to that cluster and will group it.

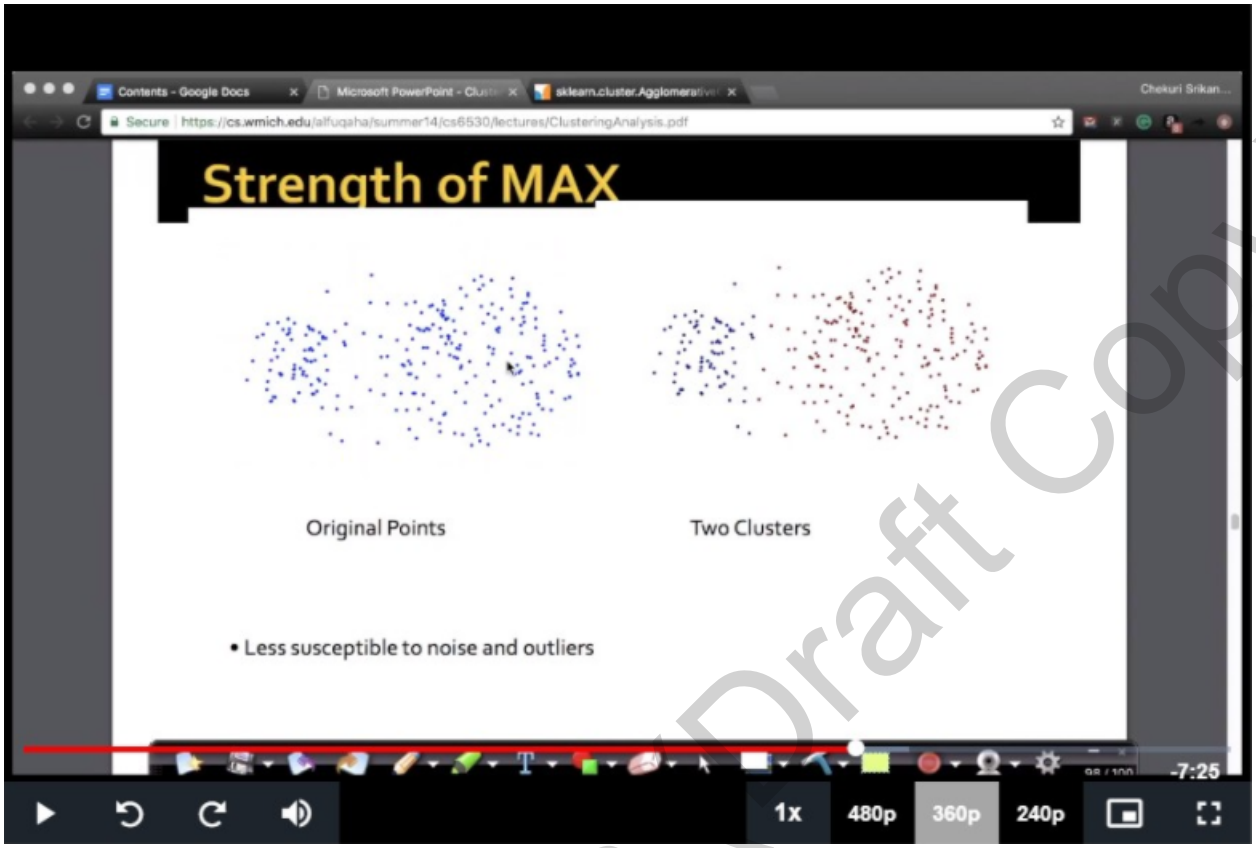
Let's consider an example for **MAX**



Timestamp : 13:58

We will do the same thing as we did in MIN, but here we use the similarity formula for similarity we defined earlier for MAX.

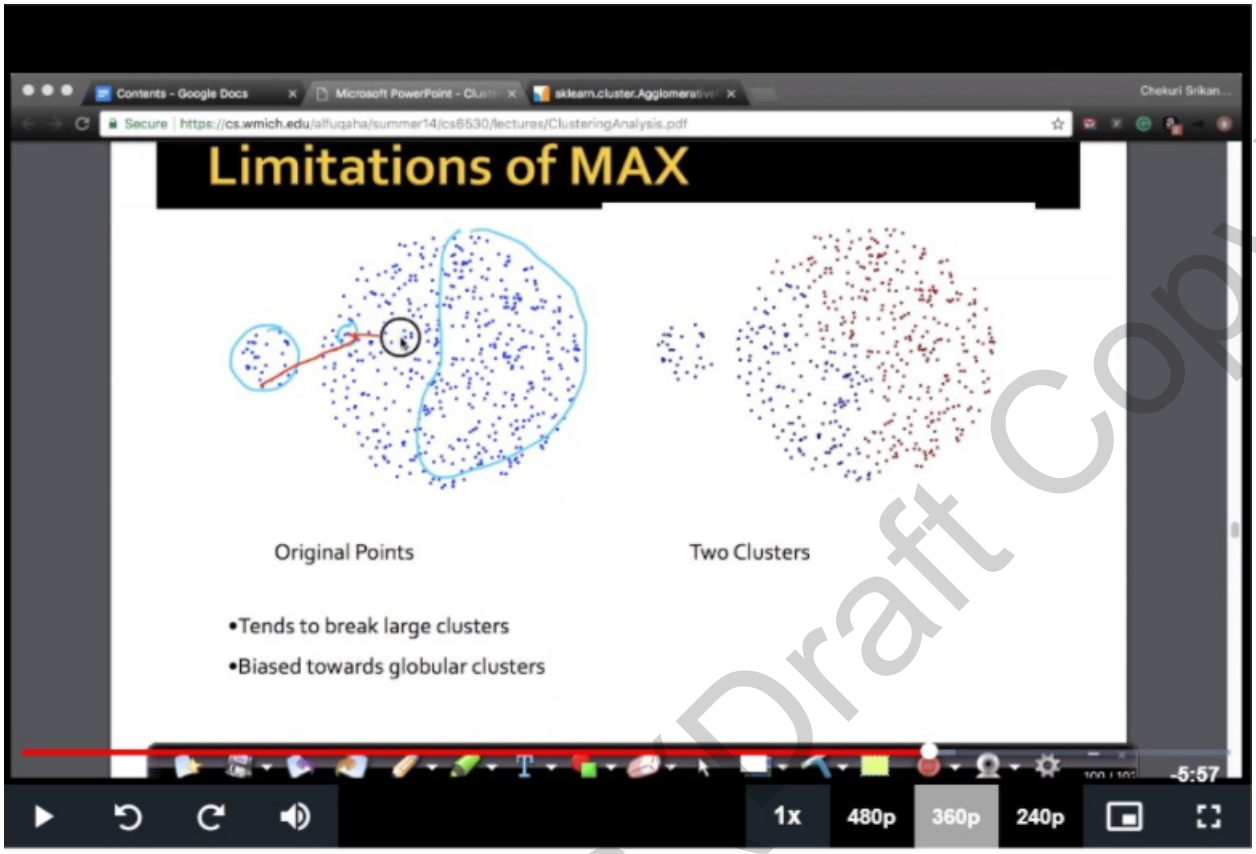
Strength of MAX :



Timestamp : 16:30

1.) It's less susceptible to noise and outliers. Even Though the noise points are closer to some of the clusters, since we are considering MAX, it is ignored.

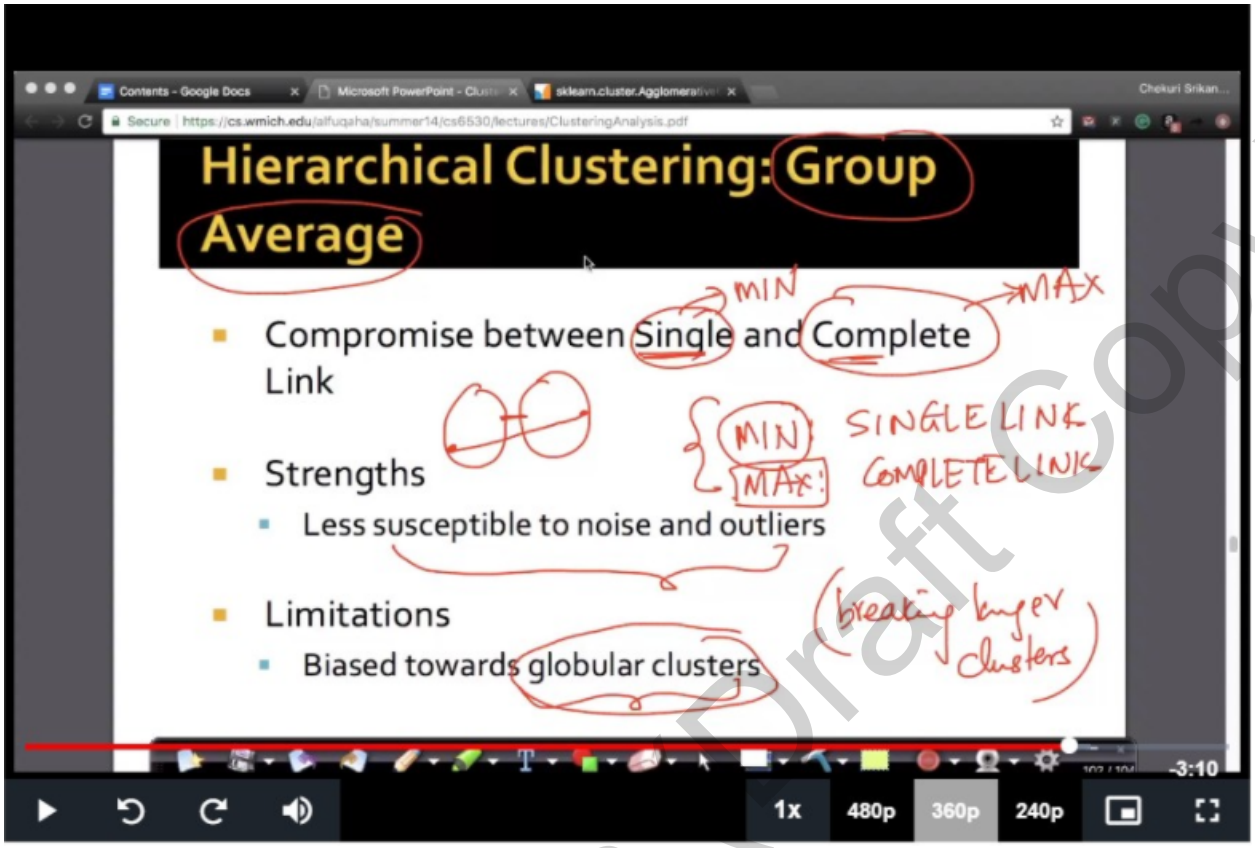
Limitations of MAX



Timestamp : 17:58

1.) Tends to break large clusters.
2.) Biased towards globular clusters.

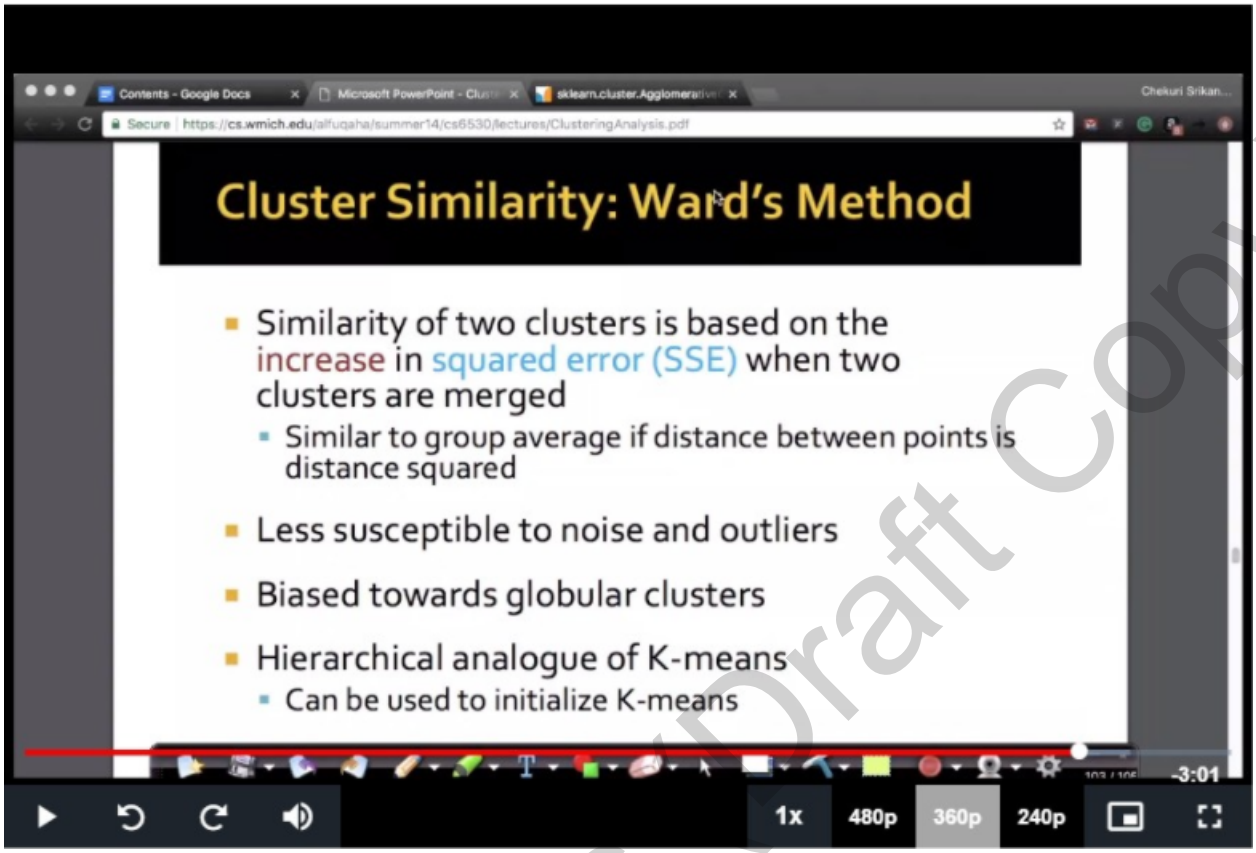
Group Average



Timestamp : 20:45

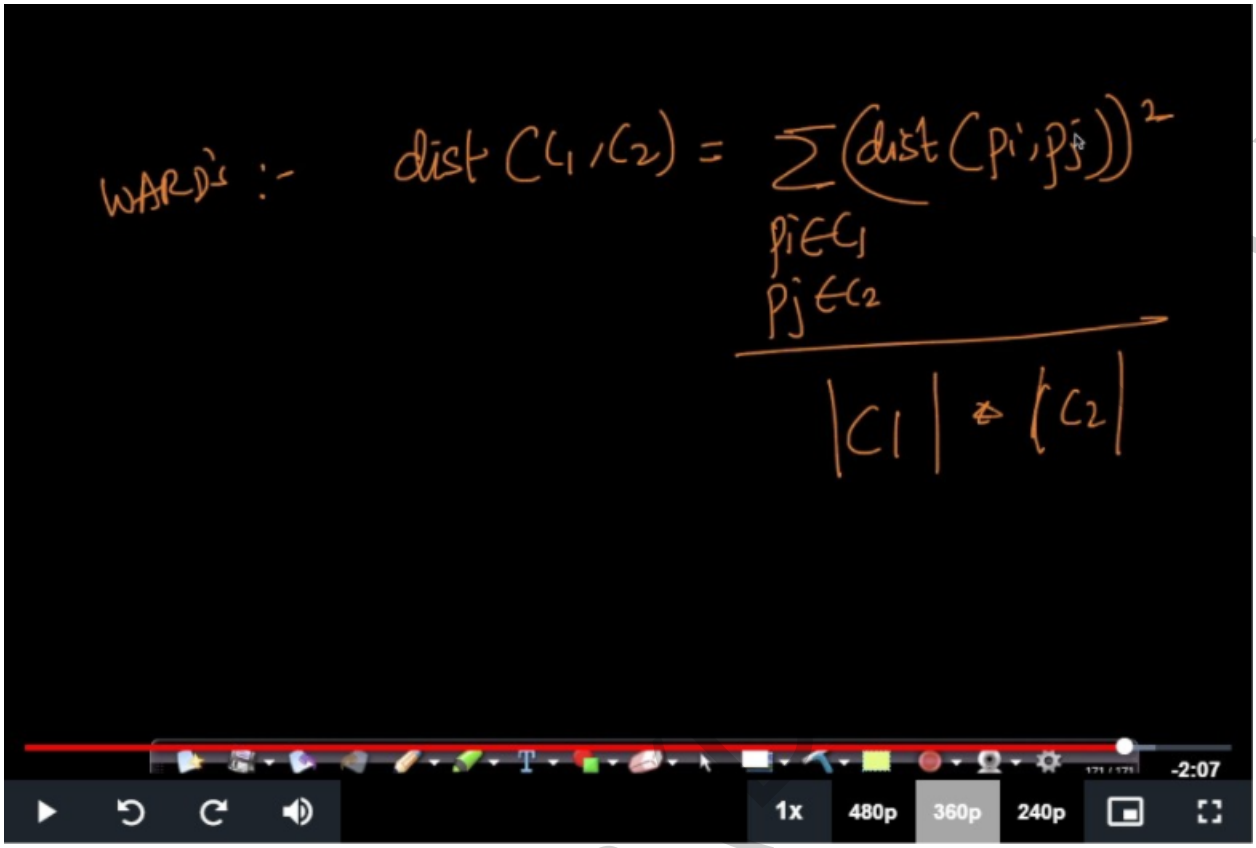
It's a combination of both MAX and MIN. So, both strengths and limitations are shared. It's biased towards globular clusters i.e, if there is any symmetrical shaped cluster or any shape , then the algorithm might group it . But in reality, it may not be the case.

Ward's Method



Timestamp : 20:54

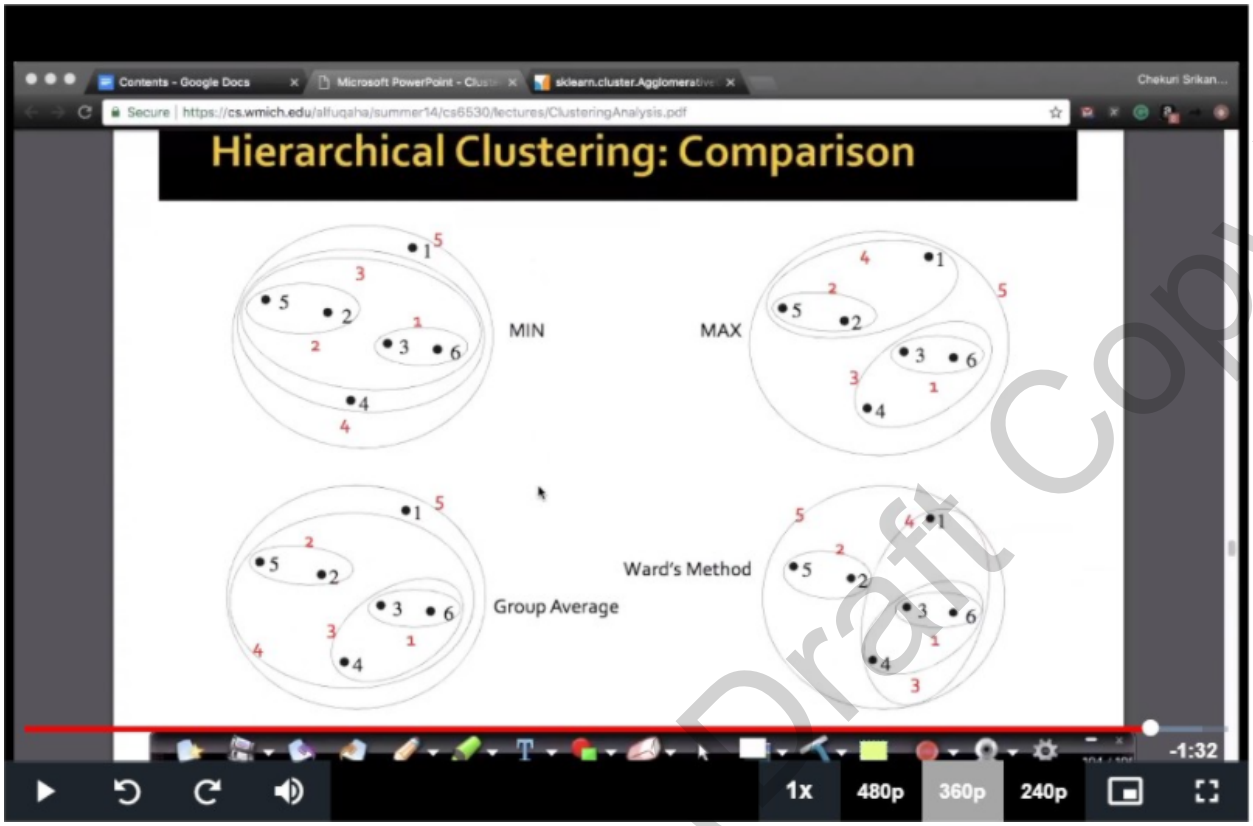
1.) It's similar to group average, if the distance between points is distance squared.
2.) Less susceptible to outliers.



Timestamp : 21:48

3.) As you can see, we are simply squaring the distance.
4.) Biased towards globular clusters.
5.) Can be used to initialize k-means.

Finally, we can make a comparison between all the methods.



Timestamp : 22:23

51.4 Time and space complexity

Space complexity :

O(n^2). This comes from the fact that we need to store the similarity matrix. If there are n points, then the shape of the similarity matrix is n*n or there are n*n entries. So, the space complexity is O(n^2).

Time complexity:

O(n^3) .

1.) There are utmost n iterations.
2.) At each iteration, we group 2 clusters.
3.) We also need to update the similarity matrix. - O(n^2).

So, it's roughly O(n^3).

We can optimize certain operations by using some advanced data structures. We can reach O(n^2*log(n)).

The time complexity is quite high. N is usually large in ml problems. We may have a million records. So, O(n^3) is large.

As a final conclusion, an agglomerative algorithm is not useful if n is large.

51.5 Limitations of Hierarchical clustering

1.) There is no mathematical objective that we are directly solving. For k-means, we had a very clear mathematical objective. We are basically using methods like MIN,MAX,AVG rather than defining an optimization function and then optimizing it.
2.) Methods like MIN,MAX,AVG,etc have their own limitations.
3.) The time complexity is O(n^3) which is not feasible if n is very large.

51.6 Code Sample

As the video lecture 51.6 is only about the code sample discussion, we are not providing any notes for it. For any queries regarding the code samples, please feel free to post them in the comments section below the video lecture (or) you can mail us at mentors.diploma@appliedroots.com