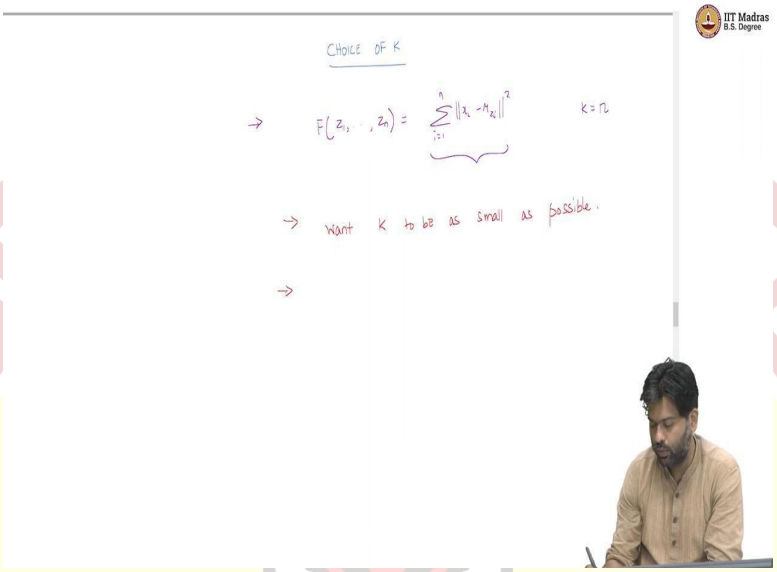
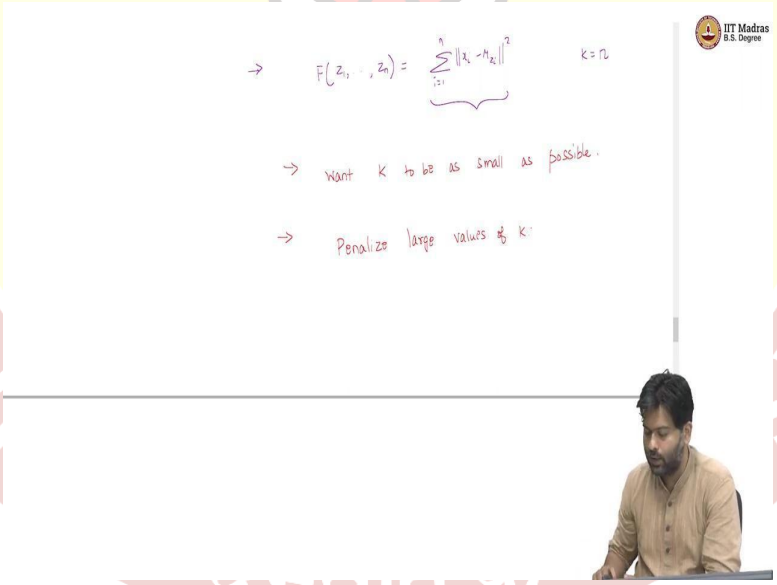
IIT Madras

ONLINE DEGREE

Machine Learning Techniques
Professor Arun Raj Kumar
Department of Computer Science and Engineering
Indian Institute of Technology, Madras
Choice of K

(Refer Slide Time: 0:13)





So, the next thing we are going to look at with respect to the Lloyd's algorithm is how do we choose K? In practice, you are not going to be given this K, you are just going to be given a bunch of data points. And typically, you do not know how many natural clusters are there in the dataset. So, which means we need to come up with a method to automatically figure this out.

So, what I am going to give you today is some broad principles as to how you can potentially get this choice of K, the specific methods, who will perhaps see it in a tutorial session. But what I wanted to just show you is a high level idea. And, all of what we will see are typically heuristics. So, and there are several other heuristics also, which we would not be able to cover completely in this short time. But the way to think about this is similar. So, that is what we will try to understand.

So, how do you choose K? So, let us first ask the question, what happens? So what is the goal? So, we have an objective function, which you want to minimize, which is $F(Z_1,\ldots Z_n) = \sum_{i=1}^{n} \| x_i - \mu_{x_i} \|^2$. This is our standard function that we have been dealing with so far. Now, I can choose any K for this potentially, let us say I choose K as n. But n is the number of data points, then how does this objective behave in that case? Well, if you have the number of clusters equal to the number of data points, then it means that every data point is its own cluster.

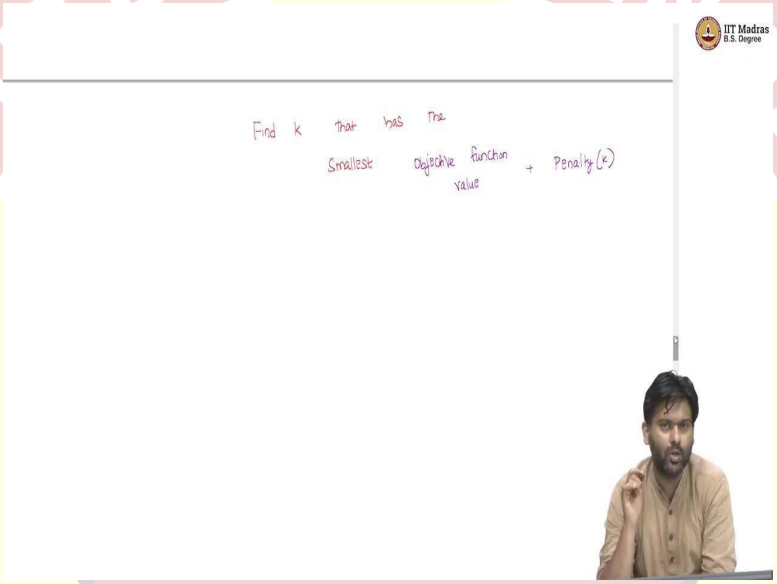
So, then what happens to this objective? You can pause and think it is not too hard to see that this objective will simply be 0. Which means though, we want to minimize this objective, that is for a fixed K, if you want to allow for any choice of K and ask which choice of K minimizes this objective, then naturally K equals n would do that, because that would give you 0, and that would be the smallest that this objective can take. So, that does not really make sense. Because we do not want the n clusters for our data points. We do not want to say that each data point is a separate cluster. I mean, that is not really much of clustering is it?

So, which means what we want is, you know, we want K to be smaller, we want to understand the data in a compressed form, which means we want K to be smaller. So, making K larger will reduce the objective value, definitely. But then that is not enough for us. So, that is not the goal, the goal is not achieved there.

So, we want K to be as small as possible. But does not mean that we are completely letting go of the objective function also. So, because for a given K, we still want to find that partition that minimizes this objective or gets close to minimizing this objective as much as possible. That is our goal. But to choose K, we should not only rely on this objective function that is the main thing. If you only rely on this objective function, then we will just end up with K equals n, which is useless. We want K to be small. So now, how can we do this?

One way to do this would then be to say that, well, I saw who can run this algorithm for multiple Ks, but then there are two different things that I need to measure. One is, if I run the algorithm with a specific K, I get a partition as the output of the algorithm. How, what is the objective value for this partition? That is something that I can measure. But that is not just enough. Now, I do not want larger Ks, I want smaller Ks, which means what we also want to measure is, how big is this K? In other words, what I am saying is we need to somehow penalize large Ks, large values of K and this is the basic idea.

(Refer Slide Time: 4:53)



So, we want somehow a K. So, broadly put, we want to find a K that has the smallest not just the objective value, because if I only rely on that, then I know K is n, objective function value, it is important, the objective function value is also important. But then it is not just the objective function value, we also want to introduce some kind of penalty for K.

It is just a function of K and depending on what your function is, you are kind of thinking of how much, you are penalizing for asking for one more cluster, you can think of it that way. So, if I move from K equals to 5 to K equals to 6, I might reduce the objective function by a certain amount, naturally, because I have more clusters. So, points will get the cluster sizes will typically become smaller, and so the distance to the means will become smaller.

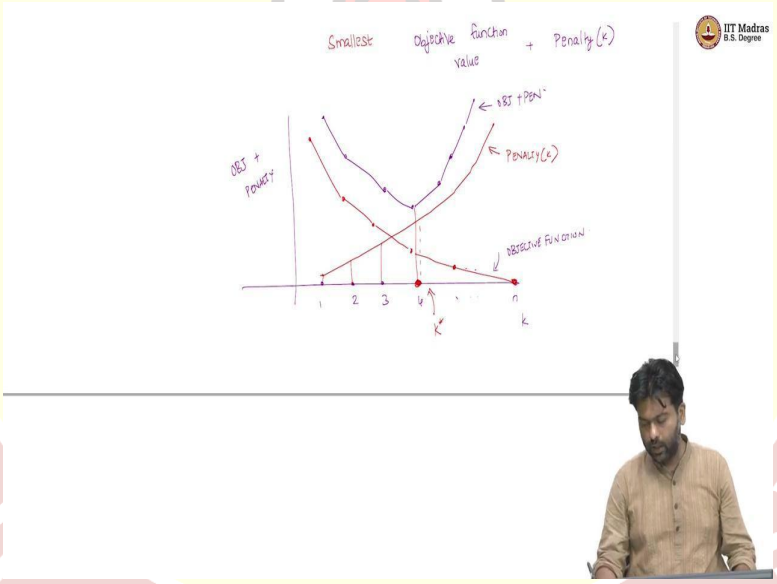
So, my objective function might reduce, but I am paying something for adding one more cluster into my data set. So, from going from 5 to 6, I have to pay some penalty. So, which is equal to penalty of 6 minus penalty of 5. So, that is the extra penalty that I am paying to

purchase a cluster, if you will. But if that is that purchase worth it, well, it is worth it if the objective function drops by a lot.

For example, if you start with K equals 1, that objective function might be really large, because there is no real clustering happening, you are thinking of all points as a single cluster. But then, the moment I make one extra cluster, of course, I am paying something for this extra cluster. But that payment is worth it, because the amount of reduction that I get in the objective function is large. But then as I keep adding more and more clusters, I have to pay more and more.

So, because my penalty is going to increase with the amount of K, but then the amount of decrease in the objective function is not going to be that much. And so there is going to be a balance that I will strike for some choice of K. That is the basic idea.

(Refer Slide Time: 7:12)



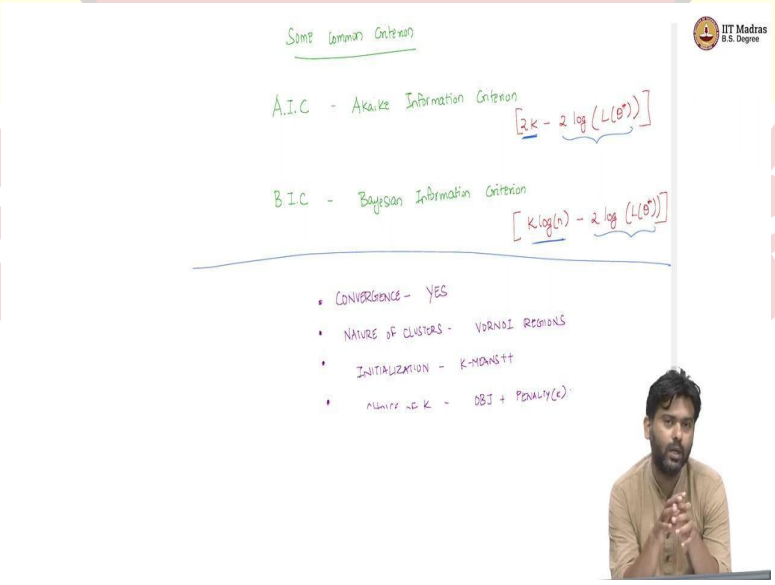
So, you can think of, if you want to think of this as a plot, it is going to look something like this. So, let us say the x-axis is just K as one K equals $1, 2, 3, 4, \ldots, n$ if you will. Now, if I just compute the objective value, well, that is going to kind of go down, so it will go down with my, as I increase my K, and it will, in fact, treat 0 when I hit K equals n. I mean there is no interpolation happening here, I am just showing this picture, the case discrete and just showing this so that the trend is visible. So, this is the objective function. Now, penalty of K is up to us to decide, so we can choose any penalty we want, we can choose something like this, maybe this is a penalty function that we want to pick for K penalty of K.

So, this gives a very low penalty for K equals 1 and as penalty increase, as K increases, the penalty increases. Now this of course, now what I am going to plot is objective plus penalty. If I plot that, which is what I care about, so then that is going to look something like this, it will go down and then after a point the penalty will take over and then it will start increasing. So, this is my objective plus penalty.

Now, which means there will be some choice of K^*, where the sum of the objective on the penalty will be as small as possible, and that K^* will be my choice of K. So, the only thing so basically, this means that I have run it for different choices of K for each choice of K, you can try out different initialization you can get, you can run your K means++, doesn't really matter.

Whatever you, whatever we want to do, or you can do uniform initialization, run it multiple times. But then finally, the best partition that you get for that K. The objective function corresponding to that is what we are considering here. And you run it for different choices of K and at some point you will find a K where this objective plus penalty is as small as possible. So, and then you choose that choice of K^*. This is one way to think about how to choose K. Do not think that I have kind of left unspecified is this penalty, penalty function itself?

(Refer Slide Time: 9:55)



So, this objective plus penalty in general people look at different criterion for choosing this, some common criterion for choosing these are the following. One is called as the AIC criterion and one is called as the BIC criterion. So, this is Akaike information may be seeing

the name, perhaps incorrectly. But this is a Bayesian information criterion you will perhaps see in more detail some of these criterion, at least in some more detail in the tutorials. But this criterion just says that you look at a function like this $\left[2K - 2log\left(L\left(\theta^{*}\right)\right)\right]$. And they will tell you what that means at a high level now, and then we will see that later in more detail. This says something like $\left[Klog(n) - 2log\left(L\left(\theta^{*}\right)\right)\right]$.

So, in some sense, these methods are giving you some way to, you know, quantify goodness of your partition by using two things like how we discussed. One is the objective value somehow has to come into picture, the other is the dependence on K itself. So, with respect to dependence on K, one criteria, the Akaike information criteria, says it has to go linearly with K. The second one also says linearly K, the Bayesian information criteria, it says K times logarithm of n. Again, so these criteria are optimal for different ways you assume as to how the data itself is generated in a probabilistic fashion.

So, so far we have not talked about data being generated in a probabilistic fashion at all. So, it might not, this might not perhaps be the best place to talk in detail about what these criterion are. Nevertheless, I just want to say at this point that people typically make some assumptions about how the data is generated and then try to argue if this assumption is true, then the best choice of K should be something that minimizes this quantity, which depends on K, which is okay. But then the way it depends on data is via a slightly different way of looking at the objective function, which is because we assume more things about how the data is generated, you can do something much more nuanced by asking what is called as the likelihood of seeing this data given some parameters that you would estimate from data and so on.

At this point, we won't dwell deep into this because we have not spoken too much about estimation, problem of estimation, which we will talk about later. But when once you understand estimation and how it fits into the broader context of, in this case, model selection, then we will perhaps revisit this and then try to tie things together where you will understand this in a much better way. But if you have already understood likelihoods and probabilistic way of generating data, then you can think of this as the objective function appears in terms of a likelihood term here and then the dependence on K of course, is linear, as you increase K, then you pay more and more price in a linear fashion.

In one case, it is just simply depends on K and other case it also depends on n. Again these are not just the set on stone. So, there are several other things that people use something

called as an elbow method and there are other choices are also prevalent in practice to choose K. Again, the goal is not to completely look at all these methods, but then just to give a flavor for how people typically do this. Usual idea is run it for different sizes of K and then use some criteria which balances objective one penalty to pick the best. So, with that, we have kind of completed our the main discussion about the Lloyd's algorithm and I will summarize this and that will be the end of the clustering part of our course.

So, for the Lloyd's algorithm, we put down four different questions. One question was about convergence and the answer to this is yes, the algorithm does converge. But it might converge to a local optima. The second question is what were the nature of clusters that the algorithm produces. And the answer to this we saw is basically voronoi regions. And I also mentioned that well, you can make this clustering algorithm work using a kernel trick, where you might get voronoi regions in a high dimensional space and so on.

The third point is initialization. We looked at one interesting way of initialization, which is called as the K-means++ algorithm, which does initialization in a much more nuanced way, which can lead you to some kind of guarantee about how good the algorithm itself is. And finally, now, we have seen the choice of K and the broad idea here is that you look at objective plus some penalty for K and then pick that K that minimizes this. So, with this, we have come to the kind of end of our discussion about clustering algorithms.

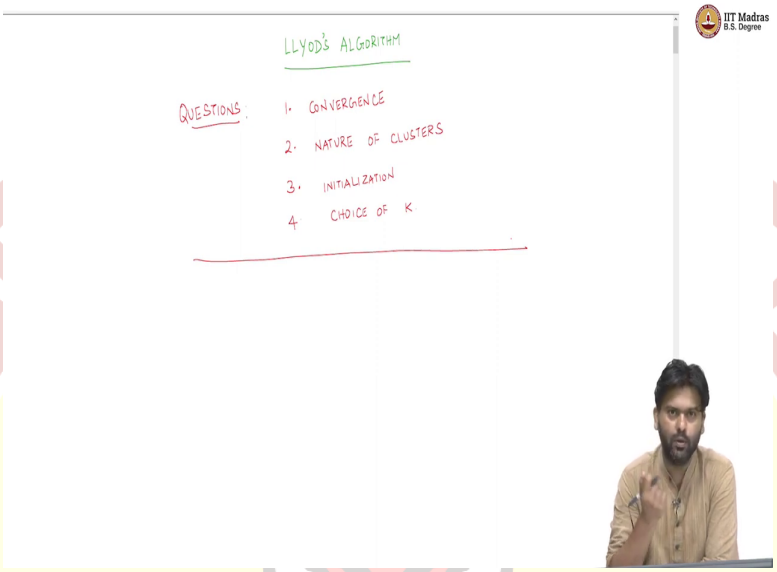
So, broadly, we have looked at non supervised learning so far in the course, we are at a place where you looked at representation learning, we also looked at clustering and some examples of popular clustering algorithms. We will continue a little bit our discussion about unsupervised learning, but then using certain method called as an estimation next time, which will not only be applicable to unsupervised learning, but then we will see later on is also applicable to supervised learning, but then it kind of acts as a bridge. And that is what we will see the next time. So, with that, we will end this discussion and I hope to see you all next time soon. Thank you.

IIT Madras

ONLINE DEGREE

Machine Learning Techniques
Professor Arun Raj Kumar
Department of Computer Science and Engineering
Indian Institute of Technology Madras
Convergence of K-means algorithm

(Refer Slide Time: 00:13)



Hello, and welcome back, we are looking at the unsupervised learning specifically the problem of clustering. And last time we put down a specific algorithm called the Lloyd's Algorithm, or the k-means algorithm for this problem. And we try to understand intuitively what this algorithm does, the algorithm, if you remember, it is a simple algorithm, you start with an potentially arbitrary partition of the data points into clusters.

And then every point looks at the distance of the point to its own mean, and compares it with the mean of other clusters. And if it finds a cluster whose mean is closer to it in terms of distance squared, then the distance squared to its own mean, then it jumps to that cluster. And this happens for every point. And that is how a reassignment of points to partition or clusters happen. And we argued that we will do this until convergence.

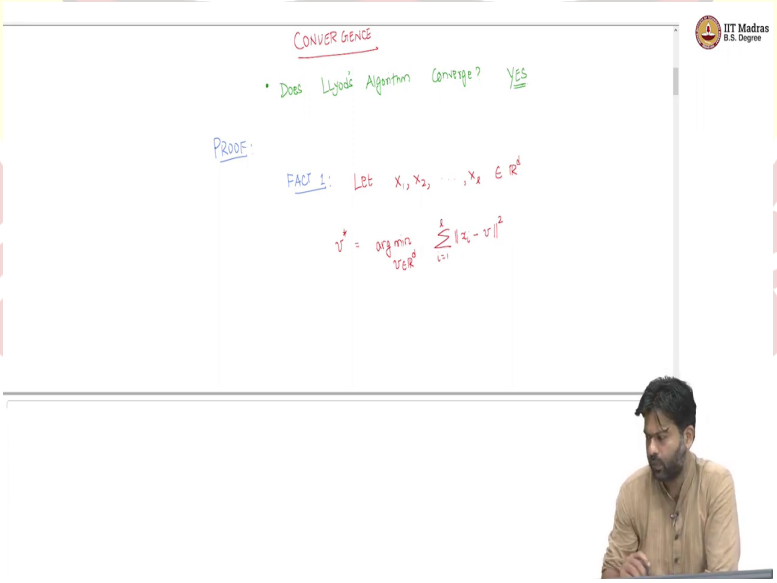
Now, what does convergence mean? Here is a question that we did not answer at that point. And that is one of the questions that comes up about the Lloyd's Algorithm which we need to answer. There are a couple of other questions also, for example, what kind of clusters does this algorithm produce? And what type of initialization should we do to get to reasonably good clusters?

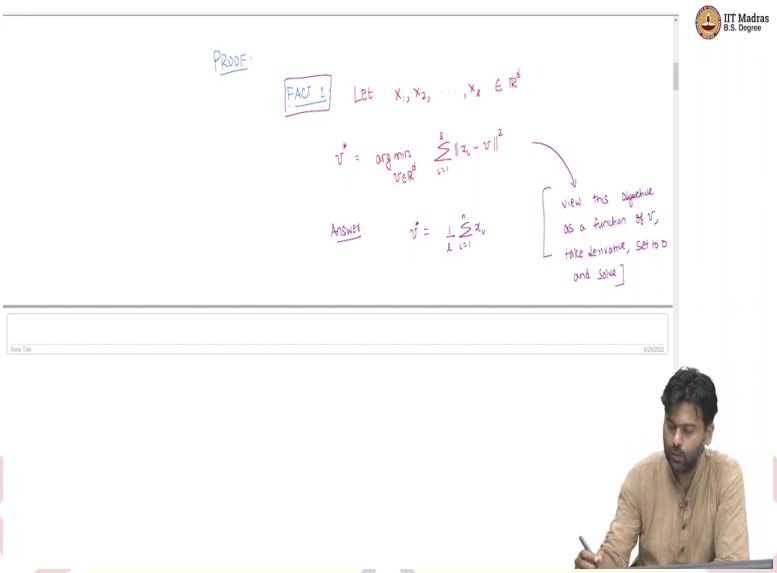
And finally, how do you choose K, because we are only given the data points, nobody tells us that there are K different partitions or clusters in this data. In some cases, it might be natural in the problem that you are trying to solve, in some cases, it may not be, for example, if you are trying to partition if you are a teacher who is trying to partition students into groups based on their performance in exams, and so on. And then you want to decide what grade to give to students.

Now, you know that there are only a set of grades S, A, B, C, D, for instance. And now you want to divide your students based on their marks in different exams into one of these buckets where you know the bucket size, or the number of grades on K, in the K-means algorithm. On the other hand, there might be cases where you do not know K, you are just given a bunch of points.

And then you want to figure out if there is any pattern where we don't know K, and then we need to come up with a way to figure out what is a good K. So, there are these four points, or these four questions that we will try to answer one by one as we go along in this course. So, the first question is the question of convergence. So, let us start with that question. Which is the question of convergence.

(Refer Slide Time: 02:57)





Does the Lloyd's algorithm converge? I did say last time that the algorithm does converge but we need to argue why. And we will do that right now. So, and understanding this argument also will tell us what kind of partitions that we will end up getting. Converge. So, the answer is yes. But then it needs an argument. So, to argue this, let us do the following. So, here is the proof. So, we will start with a simple fact.

Let us call this fact 1, which will be useful for our proof, because we are looking at data points and distance to other points and so on this proof is based on simple fact. Let us look at what the fact is. So, let us say you have a bunch of points, $x_1, x_2, \ldots, x_l \in R^d$, some set of points.

And now you want to find out, let us say, that point, which I am going to call as v^*, that minimizes over all possible vectors v in the dimension, the distance, the sum of the distance squared, i equals 1 to l, or the average of the distance squared does not really matter if it is sum or average, as we will see, to the bunch of points. This is a Euclidean distance squared.

And what we are asking is that you have a bunch of points and you want a v, which minimizes the sum of the distance squared, or the average of the district square average is just a scaling of the sum, so the minimizer will not change. We can ask either for the one that minimizes the sum or the average, in this case, let us say it we will look at the average. And we will not understand which one does this or even sum.

So, let us give it a sum, that is easier. Because we defined our partition function originally as sum. So, let us go on with the sum. So, what do you think this v would be? So, I am going to

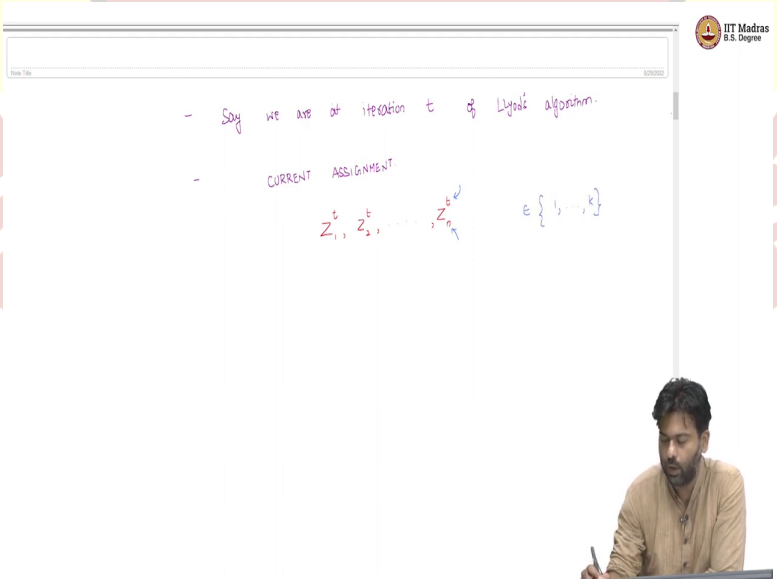
argue what this v^* which minimizes this would be, but then it might be a useful exercise to pause for a bit and think about what might be the answer to this problem. So, the answer to this is, v^* is just the mean of the data points.

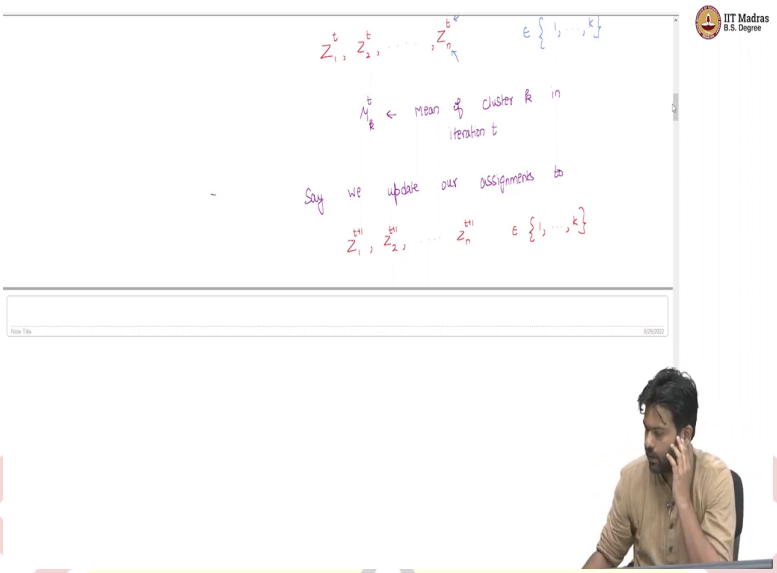
We have a bunch of data points, and then you are asking which point minimizes the Euclidean distance squared to all the data points, the sum of that is minimized by which point, well, that happens to be the mean of this bunch of data points. So, well, you can view this objective as a function of v, take derivative, set it to 0, and solve. So, that is how we would get this, I will not go through the steps. It is a very simple derivation.

And that is why I am not doing it. So, I would strongly encourage you to try this out. Especially if you are not so used to taking derivatives with respect to vectors. So, those would be like, not just a single value, the derivative would be like derivative in each direction for each component, so that would be like a gradient vector.

And then you want to treat this as a function of v, take the gradient with respect to v, set it to 0 and see which v solves that, and you would see that it would be the mean, that is a useful exercise to do. For us, it is just a fact, we will hold on to this fact, we will use this later on when we need it.

(Refer Slide Time: 07:11)





So, now we are going to prove why the Lloyd's algorithm actually converges in a very nice way. So, let us say we are at some iteration, let us call this some iteration t of the Lloyd's algorithm. Now, let us say our current assignment of points to clusters looks as follows. So, let us call the current assignment $Z_1^t, Z_2^t, ..., Z_n^t$, where of course the t corresponds to the iteration number, and the subscript corresponds to the particular data point. Remember, all of this are between 1 to k, each of these takes a value between 1 to k, because these are cluster indicators in our notation.

Now, let us also define μ_k^t as the mean of cluster k in iteration t. So, once you have a partition and the t-th term, there is some partition we are currently at, and each point has its own cluster indicator variable. So, you can basically the Zs will tell you how the data points go into buckets. And each bucket you can compute the mean. And let us call that mean as μ_k^t.

Now, what happens next is, well, the algorithm does a reassignment. We are trying to prove convergence of the algorithm. So, let us say at step t, we assume that the algorithm does not converge. Well, if the algorithm converges in step t, then there is nothing to prove. So, because you are attempting to prove that the algorithm converges, so let us assume the algorithm does not converge and see what happens.

So, there is some reassignment, which means at least one point is not happy with the distance squared to its own mean in its box, but then it has found a different box with a mean, whose

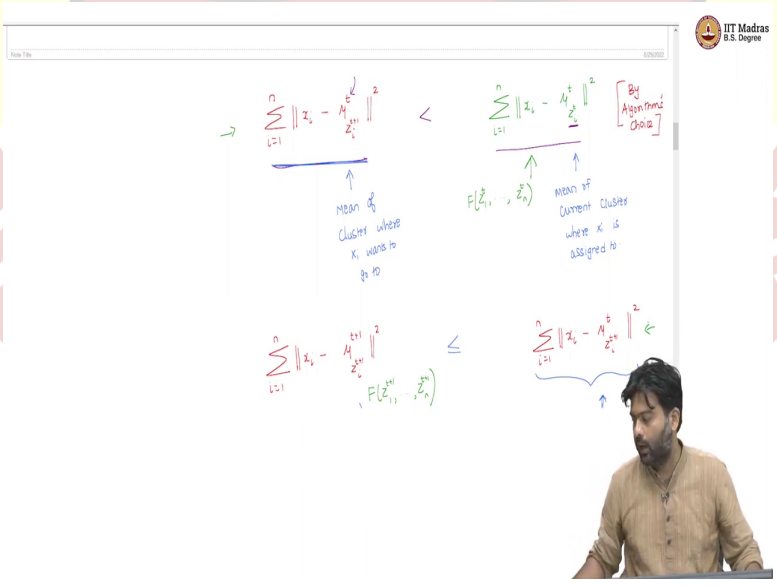
distance squared to the point is closer, strictly closer than the current mean. So, that is when a reassignment happens.

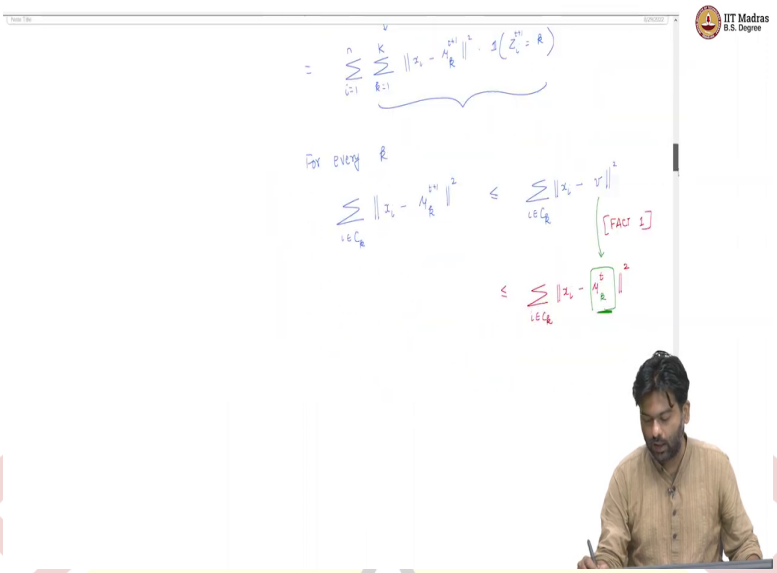
So, let us say we update our assignments to the next steps assignments. Now, these would be $Z_1^{t+1}, Z_2^{t+1}, ..., Z_n^{t+1} \in \{1, ..., k\}$. But then things have changed. So, points have moved around. The question is what can we say about this update? So, when we move from one partition, which is given by Z_1^t to Z_n^t to a different partition, which is given by Z_1^{t+1} to Z_n^{t+1}, what can we say? Is this a good change? That is what we are trying to capture.

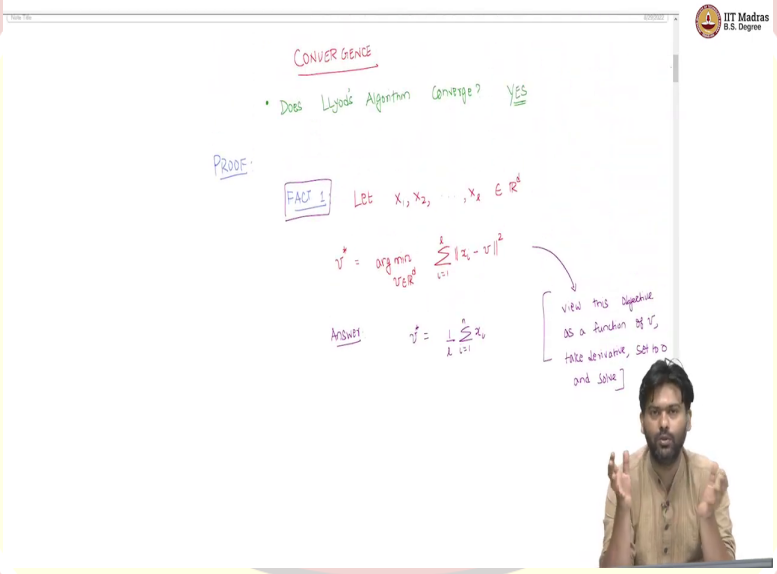
So, you are in some way of putting points in boxes. Now, you are moving points around. Now, after this the partition that results, is it a better partition than the previous partition? So, in some sense, that is what we are trying to understand. So, how do we argue this is a better partition?

Well, we have to look at our objective function, we had an objective function which given a partition will tell you how good that partition is by assigning it a number. And then we said that we wanted the smallest value for that objective function, the partition that gives us the smallest value. So, in terms of that, what can we say?

(Refer Slide Time: 11:06)







So, let us take a look at this. Let us say I want to consider this particular expression, $\sum_{i=1}^{n} \|x_i - \mu_{z_i}^{t+1}\|^2$. And then I want to compare it with a different quantity. And then we will talk about what these quantities are in a minute. So, this is $\sum_{i=1}^{n} \|x_i - \mu_{z_i}^{t}\|^2$. So, I have put down two quantities. Both of these are sum of distance squared of points to some other vectors, some other means.

But what is, let us look at this first. This is simpler thing to understand first. So, what is this quantity? Well, we are in the t-th iteration. Every point is being measured in this expression

to the mean of the box to which it is assigned to. You are computing the distance squared of every point to the mean of the box to which it is assigned to. So, x_i is assigned to the box Z_i^t.

That is by definition where x_i is assigned, Z_i^t is a cluster indicator of the ith data point in the t th iteration. So, the ith data point goes to the box Z_i^t, which is some number between 1 to k. So, basically, what this is capturing is just the sum of distances of each point to its own mean in the tth iteration. Now, what is this quantity on the left capturing? Well, this is doing something interesting.

So, it is trying to capture the distance of each point to the mean that it is being assigned to in the next round. So, let us be careful when I say mean, it is being assigned to the next round. Now, you are still looking at means of the tth round, you are in some partition there is μ_1^t, μ_2^t, μ_3^t, till μ_k^t, you are still measuring x_i to one of these, but which one, well, you are measuring it to the one which is in the box that you want to go to.

So, you are not measuring it every data points distance to its own mean, but then you are measuring its distance to the box where it wants to go. Well, some data points do not want to jump. So, it does not want to jump, the data point is happy with its own mean in which case Z_i^{t+1} for the data point will be same as Z_i^t. So, then it is the same mean that you are measuring it with.

But then there are some data points which might find that there is a mean which I am closer to so I want to jump and that is when the reassignment actually happens. Because the algorithm does not converge there is at least one point for which the distance to $\mu_{z_i^{t+1}}^t$. So, the mean of the box, where I want to go to is strictly lesser, the distance squared to that point mean is lesser than the distance squared to my own mean.

So, because there is at least one point with this property, and whenever a point decides to jump it is because this distance to where it is comparing itself to is strictly lesser than where it is currently assigned to. So, this whole sum is going to be less than the right-hand side. Well, this is simply by algorithm's choice. So, the algorithm makes this choice to make a jump only when it finds a different cluster where the mean is strictly closer to then the current cluster.

But remember, this quantity on the left is not the objective value of the partition that you would get in the t plus first round. Why? Because I have not made the jump it. So, this is saying I am closer to the other guy, and I am going to measure my distance to the other guy. So, that is this situation. So, whereas if you really want to measure the objective function in the next round, well, you should compare each point to the mean that you get in the next round, so after you have made the segments.

There are a lot of points which are jumping around, and after every point jumps around now you compute the mean in each box. And now each point is compared to its own mean, that is the objective function in the next round, but that is not what this is. So, this is an intermediate quantity if you will. So, let me write this. So, this is mean of cluster where x_i wants to go. And this is mean of current cluster where x_i is assigned to. That is the difference.

So, now what we really want to argue is how does the objective function change after you make the partition change. So, which means we want to see this quantity, $\sum_{i=1}^{n} \|x_i - \mu_{z_i^{t+1}}^{t+1}\|^2$, well in the t plus first round, how does Z_i^{t+1} th mean the one that x_i is being assigned to in the t plus first round, so we are comparing every data point to the mean in the t plus first round, and then we want to compare this quantities value.

Now, with the intermediate quantity. The intermediate quantity being $\sum_{i=1}^{n} \|x_i - \mu_{z_i^{t+1}}^{t}\|^2$. Now, what might be the relation between these two quantities. The quantity on the left is your partitions objective value evaluated in the t plus first round. The quantity on the right is an intermediate quantity, where after the t-th round, you are comparing each data point to the mean of the cluster where it wants to go.

So, now, how do these two values compare? What can we say about this quantity? Well, this would be a very instructive exercise to pause and think. I will tell you what the answer is and then I will argue why it is. The relation between these two terms is that the left-hand side term is less than or equal to the right-hand side. And why is this? Why should this be true? Now, instead of writing down a long argument as to why this is true, it is not really long argument, but then let us try to understand what is happening.

Let us focus on one box, one cluster. So, in the tth round, there were a bunch of points that were assigned to this cluster and this cluster had a specific mean. So, it had a specific mean. Now, there are some points which are happy with the mean, they are not changing. There are some points which are not happy with the mean, because they have found a different mean, which seems to be closer. So, those points are leaving this and then jumping out.

Now, there are some points which are remaining, but now there are some other points from different clusters, which think that this mean is closer and then they are coming here. So, now, if I look at the points of this particular cluster, after doing this reassignment, after the points, some points have gone out, some points have come in, now all those points are here. Why? Because they have been measured to the mean of the previous round, that is each x_i that is in cluster k. So, now, what does this compare it to on the right-hand side?

Well, this is comparing it to the mean of this box in the tth round. Now, what are we going to compare it against in the t plus first round, we are going to compare it against the mean of the t plus first round. So, after some points have gone out, some points have come in, now you have a set of points you compute the mean and then you are comparing the distance of each point to its own mean.

Earlier, the contribution to the right-hand side term for each point in this cluster was its distance to the previous mean of this cluster. Now, the contribution that each point to the objective function on the left-hand side is the distance to the mean of the updated cluster, so in the t plus first round. So, which means all the points in the right-hand side expression were contributing their distance to the mean of the tth round.

On the left-hand side expression, they are contributing the distance to the mean of the t plus first round. The question is which of the sum is smaller? Well, now this is where we use fact one. If you remember what we did, in fact one, we said that you have any bunch of points, does not matter what the set of points is, if you want to measure the distance or the contribution of each data point as a distance squared to a single point.

Well that contribution is smallest when the single point is the mean of the bunch of points. Which means in the t plus first round, because every point for many, if you break this expression into k different clusters, and then compare each cluster's points assigned to that cluster to its own mean, that would be smaller than the contribution of each of these points to

the previous mean, which is what these points were contributing in the right-hand side expression.

So, which means what is exactly happening is the following. So, this is less than or equal to this, well, because I can write this somehow like this, so I can write this expression as $\sum_{i=1}^{n}\sum_{k=1}^{K}\|x_i - \mu_R^{t+1}\|^2 \cdot \mathbb{1}\left(Z_i^{t+1} = k\right)$. Of course, I can write the left-hand side expression like this, because this is just, I am breaking the entire contribution of all the data points, which is what is summed here into each cluster.

So, now this is basically I can break this into a bunch of points that go into each cluster, and then see what is each points contribution to its own mean. Now, for every k, we know that, well, the set of points that belongs to that cluster, let us call that C_k. So, the data points distance to its own mean is less than or equal to the set of points, sum of distances to any v. So, this is fact 1, this was fact 1. The distance to its own mean, the sum of distance squares of data points to its own mean is less than or equal to sum of distance squared of point to any vector whatsoever.

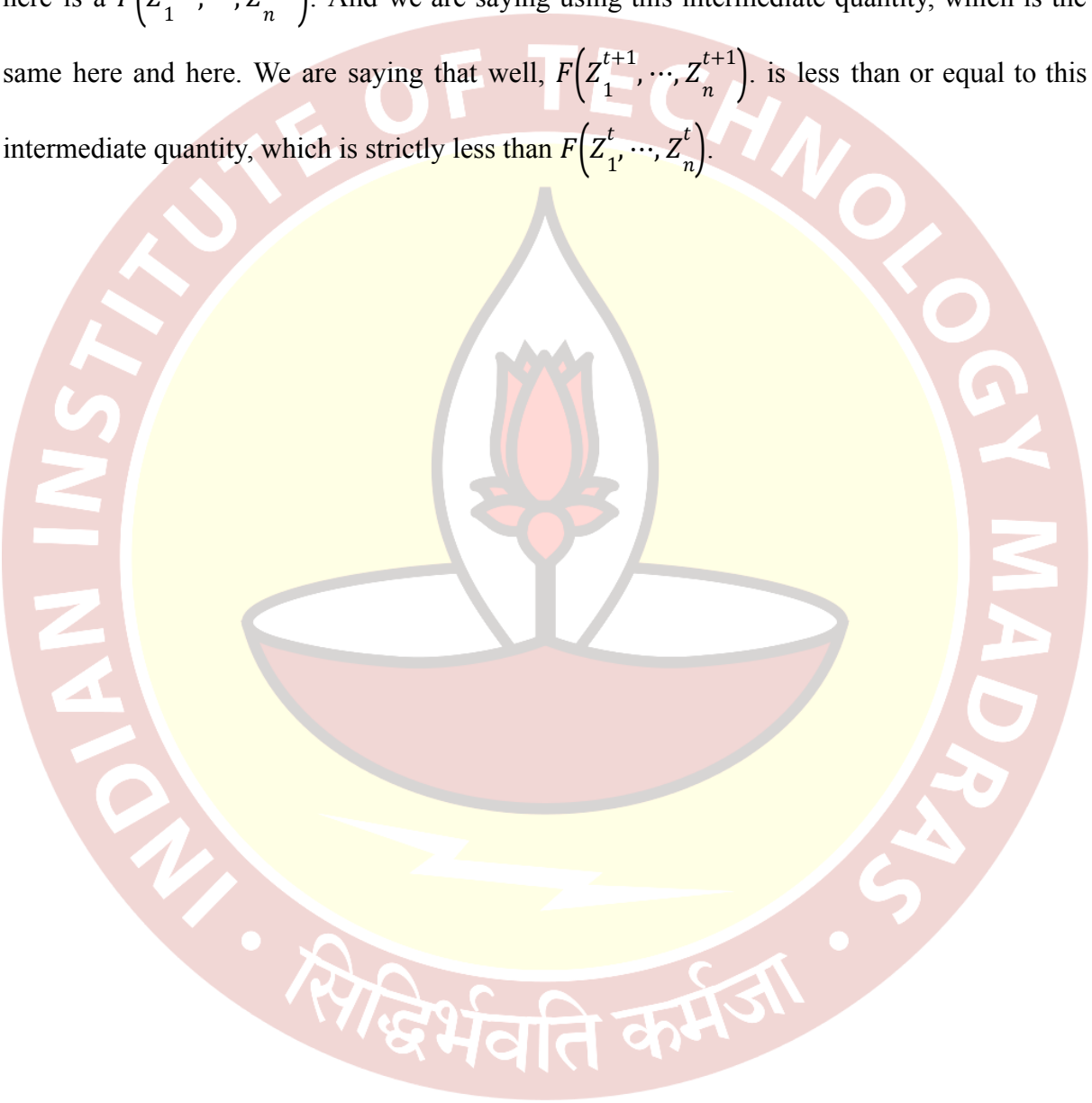
And in particular, this means that this has to be less than or equal to $\sum_{i \in C_k}\|x_i - \mu_{z_i^{t+1}}^{t}\|$. So, but then, so what is this expression? So, why am I saying I am justified in putting replacing this with v here, because all the points in cluster k were being compared, all the points in cluster k in the t plus first iteration were being compared to the mean of cluster k in the tth iteration. So, in fact, I can even say this is μ_k, that is okay, actually, μ_k.

So, this was why these points are in cluster k in the t plus first iteration. So, every point is in cluster k, because either it was compared to its own mean, and then it was happy or it compared itself to this mean, μ_k^t, and then jumped here. So, which means that, well, earlier it was being compared to μ_k^t, now it is being compared to μ_k^{t+1}. But because fact 1 says well μ_k^{t+1}'s comparison sum of distance squared should be less than the comparison to μ_k^t. So, which means this is true.

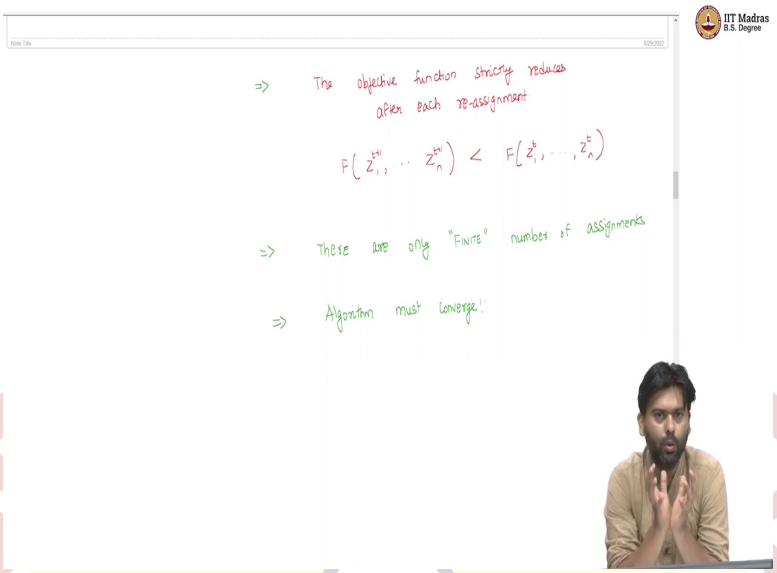
In other words, you can think as if points are moving closer and closer to each other in some sense. So, because after making this move your objective function reduces. That is what we

have kind of argued now. So, what is the argument? So, what have we gained by this? Well, we show that the objective function after making a reassignment strictly reduces, after every reassignment, that is what we have managed to show. How have we shown that?

Well, this quantity here is the objective function, this is just $F\left(Z_1^t, \cdots, Z_n^t\right)$. Now, this quantity here is a $F\left(Z_1^{t+1}, \cdots, Z_n^{t+1}\right)$. And we are saying using this intermediate quantity, which is the same here and here. We are saying that well, $F\left(Z_1^{t+1}, \cdots, Z_n^{t+1}\right)$. is less than or equal to this intermediate quantity, which is strictly less than $F\left(Z_1^t, \cdots, Z_n^t\right)$.



(Refer Slide Time: 26:36)



Which means what essentially then it says is that the objective function essentially what we are trying to argue is that the objective function strictly reduces after each reassignment that is $F\left(Z_1^{t+1}, \cdots, Z_n^{t+1}\right) < F\left(Z_1^t, \cdots, Z_n^t\right)$ if reassignment happens. Okay so, but why does this mean that the algorithm should converge? All we are saying is that we are in some partition, which has some objective value and now reassignment step happens and the objective value strictly reduces.

But why does strictly, say strict reduction in the objective value, imply that the algorithm should converge? Algorithms convergence remember means that you are reaching a specific partition where you know every point is happy with its own mean, no reassignment happens after that. That is what we mean by algorithm is converged. But why should this condition imply that the algorithm has converged?

This is a good question to pause and think. And I will answer that now. Well, in general, if an objective function keeps reducing, it could, one might imagine that well, it could keep on reducing without the algorithm converging. But now let us think about this way, what does it mean to say that the objective function strictly reduces, it means that the partition cannot repeat itself. You start with a partition it has a specific objective value.

Now, if a reassignment happens, the objective value strictly reduces, which means it has to be a different partition, it cannot be the same partition. So, the partition cannot repeat whenever reassignment happens. But there are only finite number of partitions. However large this

number could be, so, some huge number, the number of partitions of a bunch of n numbers into k boxes.

Naively, we saw that that grows exponentially, we will not really talk about the exact value of that, but then that is not the point. The point is that there are only a finite number of them, at most k^n of the m, but it is a huge number, still finite. Which means that every time a reassignment happens, I am thinking of crossing off one of these partitions.

So, because I have visited that partition, and because I cannot revisit a partition, eventually, your reassignments have to stop which means the algorithm necessarily has to converge. The finiteness of the number of partitions is what is allowing us to argue that this algorithm has to converge. There are only finite partitions, number of partitions assignments now that implies the algorithm must converge, that is the argument.

So, essentially, what we have managed to show is that every time a reassignment happens in the Lloyd's of the K-means algorithm, your objective function, which is measured as the sum of the distance of the data points to the mean of the clusters to which it is assigned distance squared, to which it is assigned, reduces monotonically, strictly monotonically.

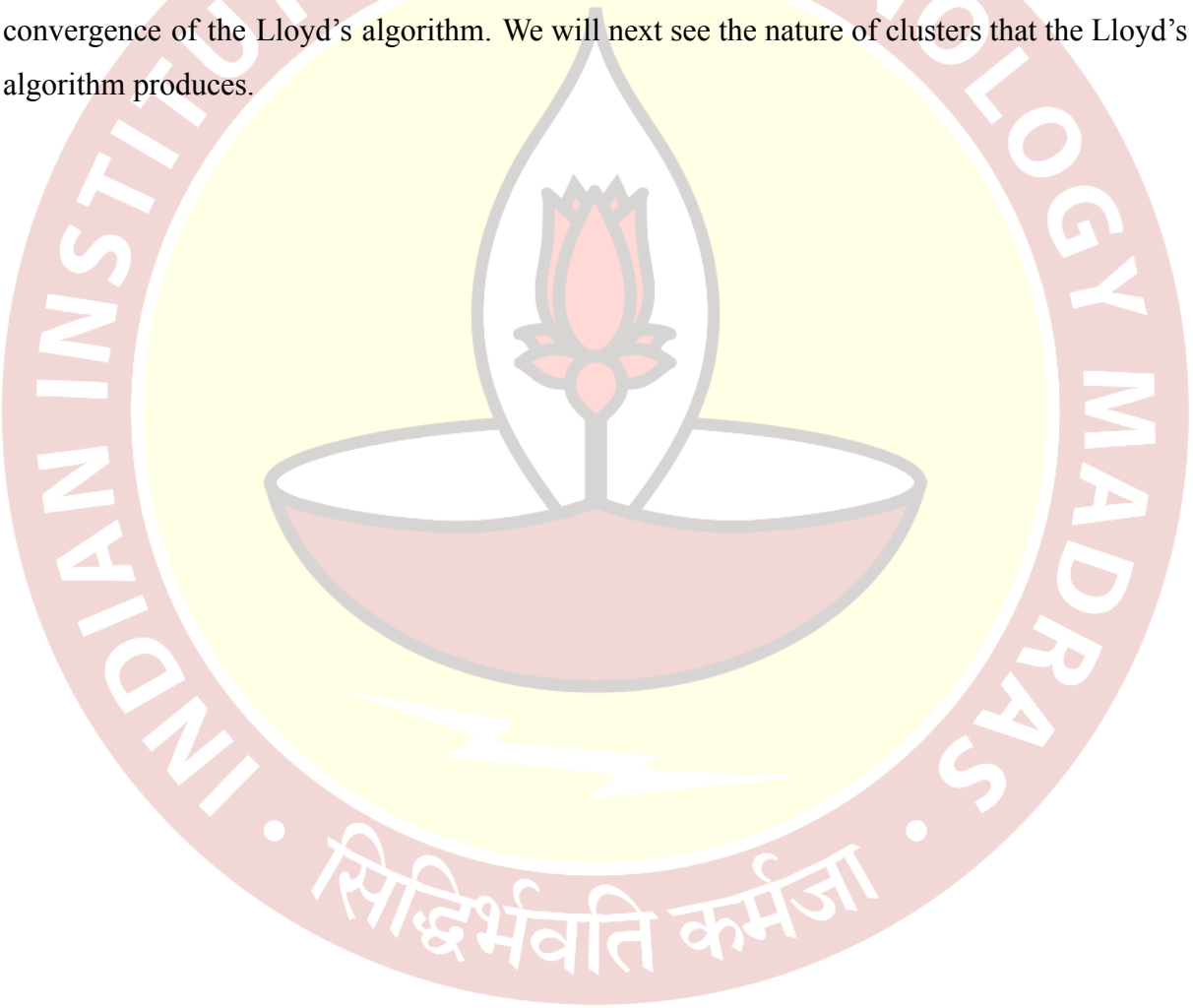
And that implies that at some point, you will hit a partition where every point is happy with its own, and it will not change anymore, because it cannot infinitely keep reducing when you have only a finite bunch of possibilities. That is the argument. Now, it does not mean that, a couple of things, let me clarify that. It does not mean that the algorithm is always going to take k^n or the number of partition rounds to converge. This does not say that at all.

So, the finiteness argument does not imply that the algorithm necessarily takes those many rounds to converge. Nothing of that sort is being implied by this. The algorithm converges really fast. So, in practice, it typically does really fast for most practical datasets. This is the worst-case analysis. Even if it had to run for k^n, order of k^n iterations, well, it has to converge. That is the argument we are making.

We are not saying how many rounds the algorithm will take. That is point number 1. The second thing is that we are not really talking about the goodness of the partitions or the clustering that will result from this algorithm. Nothing of that sort is also being said here. So, we are not arguing that the algorithm will eventually because it converges, we are not saying that it has to converge to a place where the objective function is the smallest.

That is not true at all. All we are saying is that the algorithm will converge. But it could converge to some, in some sense, a local minima. So, at that partition everybody is happy. There is no more changes that the algorithm is trying to do. But that does not mean that that partition will give you the least objective value or all possible partitions. Nothing of that sort is also implied by this argument.

This argument just says the algorithm converges, which is a very important thing to know, because otherwise, we will not be sure. If partitions could repeat, then the algorithm can get in some loop and then it will never converge. We do not know when to stop. Now, that problem is not there is what this argument shows. So, that completes our argument about the convergence of the Lloyd's algorithm. We will next see the nature of clusters that the Lloyd's algorithm produces.

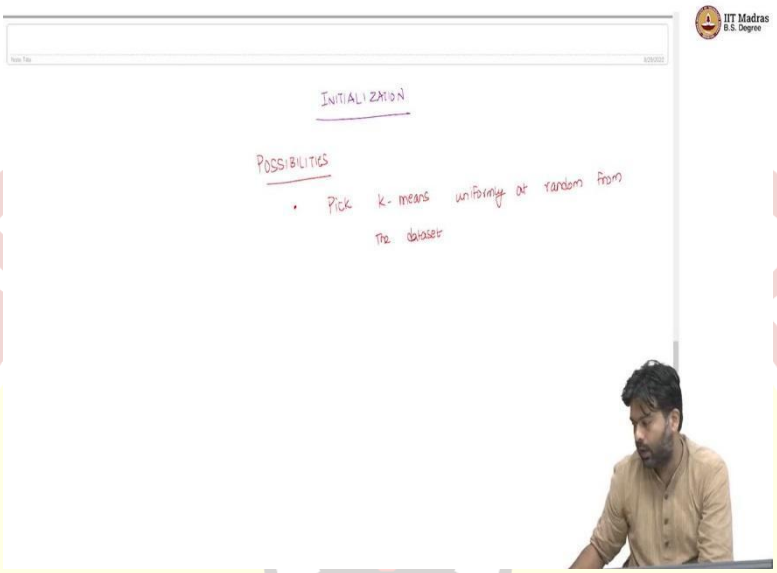


IIT Madras

ONLINE DEGREE

Machine Learning Techniques
Professor Arun Rajkumar
Department of Computer Science & Engineering
Indian Institute of Technology Madras
Initialization of Centroids, K-Means++

(Refer Slide Time: 0:14)



So, how do you initialize the Lloyd's algorithm? Well, we know that the algorithm definitely converges no matter how you initialize it, we argued that. So, our argument was agnostic to the initialization of the algorithm itself. So, we could initialize it however you want, but then some initializations might lead you to better cluster. So the question is, can we initialize it in a good way, so that we might hope to end up in good clusters. So what are some possibilities?

Well, one thing is we can simply throw data points into boxes uniformly at random, I pick a data point, I have k boxes, I will put it in a box at random. But that is not a very mindful initialization, so to say. So, we are just throwing points in boxes, it will definitely converge, the algorithm will converge, but you are not really giving that initial push to beat these local minima's, that algorithm might get stuck in when it converges. So that is so throwing points at random in boxes without really understanding the structure may not be that good an idea.

So, what do we really want, we somehow want that at the end, we know that points have the property that clusters are the property that each point is closer to its own mean. So, one thing that you can do is you can start your initialization may be in initial partition by saying that I have 1000 points in my data set, and I need to put them in 5 different boxes, I will pick 5

points uniformly at random from my data set and imagine these 5 points as my cluster centers.

It means that I am initializing, not the sets which is the cluster indicators for each data point, instead, I am initializing the means. So, I am initializing, which are going to be the means and once I fixed these means as points that are uniformly picked from the data set, now, each point will get attracted to its own closest mean, and then we will go to that box.

So, one way to do this would be to pick k-means uniformly at random from the set of data points, from the data set. So, you have 1000 data points, maybe data point number 5, data point number 72, data point number 89, 2005, 22, whatever these numbers are, those are your initial random lot, they correspond to your means.

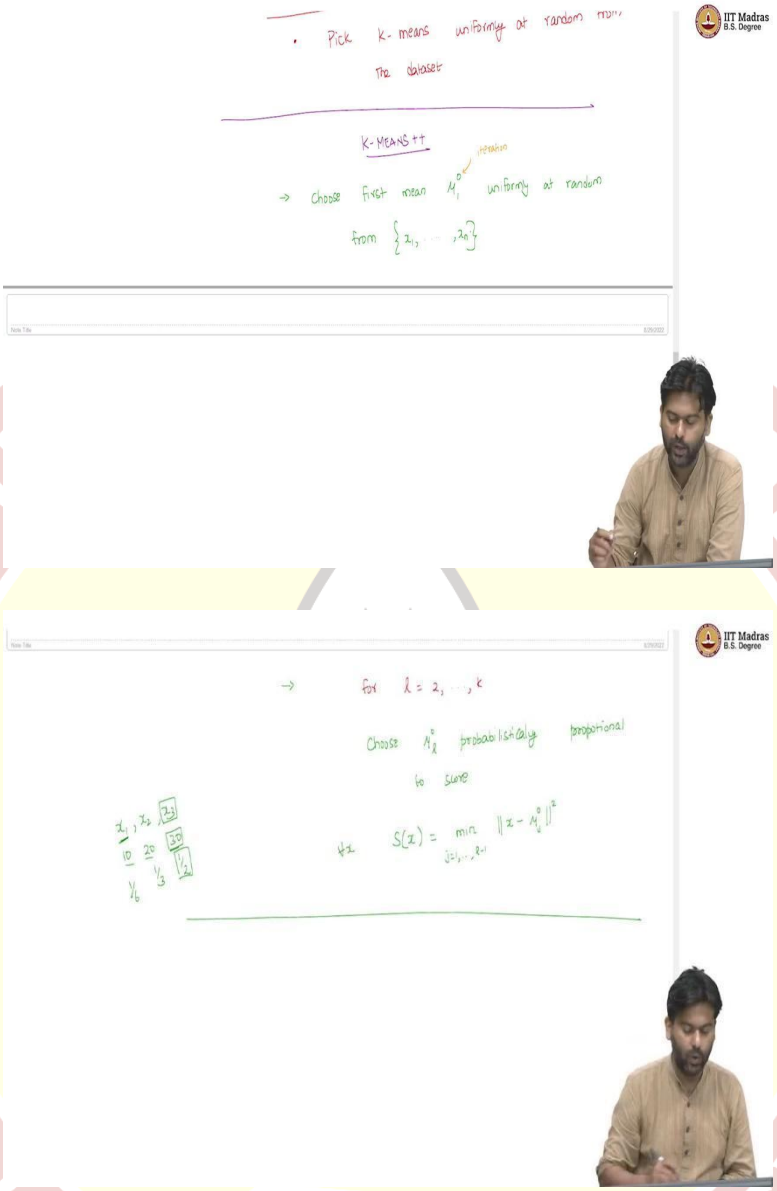
You put them in each of these boxes separately and then for every other point, you see which of these 5 points I am closer to, and then it gets assigned to that box. So that generates initialization, where at least the points have the property that they have been chosen to be closer to a particular mean, and then they go into it. Now, that does not mean that the algorithm has converged.

So, because every point is only going to a box based on the mean, which is another data point, the single data point, lot of data points might come in, and then the means will get recomputed and then the algorithm will start from there on. And so it is but then this is a very, it is not such a bad idea to do this. And then in practice, people do this a lot of times.

And in practice, this is something that is what is typically done, if you do this, and then you will run the Lloyd's algorithm, it will converge to some partition. Now, it could solve have happened that the initial k-means that you picked as points from the dataset, you might have gotten unlucky. So, what you might want to do is, you run this with different means again.

So, you again redone on the algorithm where the initial random as is again generated, it pick 5 different means k different means and then redone the algorithm. And then you do this multiple times and you end up with the pick the clustering, which gives you the best objective value. So that is what people typically a lot of people typically do in practice. So, this is one way you could do initialization.

(Refer Slide Time: 5:04)



Another slightly more principled way is what is called as the k-means++ algorithm, which is just k-means algorithm with a better initialization. Here, the idea is the following, so, we still want to pick k points from the data set, which we want to think of as the k-means to begin our data algorithm with. But if you are picking them uniformly at random, you might not you are not putting effort into picking these means carefully.

So, what do we want these means to look like at the end? So, let us say we run the algorithm, algorithm converges, if you are happy, then it means that these means are as away from each other as possible, because the means are some sense representing each cluster, we are

thinking of the entire data set has been compressed into k different representatives, which are these means. And these means how to be as different as possible.

So, they are be as away from each other as possible. If they are close to each other, then it means that when the representatives are similar, well, then why should I want 2 different clusters, I could have had single cluster? So, this idea is formalized, while you initialize. You initialize carefully such that your means are in some sense as apart from each other as possible.

But then you do it with a slightly probabilistic twist and that is what will lead to this k-means++ algorithm, which I will describe now. The algorithm does the following, so it tries to not choose k-means as k different data points from your data set uniformly at random in one shot, it does not do that, whereas it does this in an iterative fashion.

What it does is the following, so first chooses the first mean. Let us call this μ^0_1, where this naught just means that, this is the iteration number, this just means it is at initialization, μ^0_1, uniformly at random from your set of data points, x1,..,xn, you have endpoints, pick one point at random and quality your first mean. So, the first mean is fixed. Now you need to choose the second mean.

So now here, this is done in an iterative fashion. So, for l equals to 2 to k, where l represents the l-th mean that you are going to pick, you are picking one at a time. So what you do is the following, so you want to pick, you want to choose μ^0_l, that is the l-th mean at initialization. Now probabilistically, proportional to score the following score. So, you are going to give, so basically, you have a bunch of remaining data points.

So, you have already let us say, pick 5, means you want to choose the 6th mean now, you have a lot of data points, you have 1000 data points, 5 points have been already assigned as means you are in the 6th round, which means you still have 995 points from which you need to pick one. Now what we are going to do is, we are going to give a score to each of these points, and the score is a positive number.

And then you are going to pick a point with probability proportional to the score higher the score more the probability of picking that. How do you do that? You kind of can normalize this score over all the remaining points and then that will give you a probability distribution

over these remaining 995 points and then you can sample one point according to this probability distribution.

For example, instead of 995, we just had 3 points, let us say remaining points from which you need to pick one. The score of first one is 10, the score of second one is 20 and the score of third one is 30, let us say, we will talk how you get the score in a minute., let us say you had the scores 10, 20 and 30.

Now, what does it mean to say, I am probabilistically picking one point according to the scores, it means that I am going to normalize these scores to say instead of 10, 20 30, I am going to think of it as 10 divided by the sum of these scores, which is 10+ 20 + 30, which is 60. So this will be 10 by 60, 20 by 60, 30 by 60, which is one sixth, one thirds and half and that would be the probability with which I will pick each of these data points.

So, now what are the scores? Well, now what do I want to do? I want to be as away from the means that I have been that I have chosen so far. So, let us say I have already chosen 5 means I want to pick the 6th mean, so now every point is a candidate for the 6th mean. Now, how am I going to judge this point as being good or bad with respect to the 5 means that I have picked already?

Well, I will compute the score of each of these points to the each of these means that I have already seen and see what is the smallest of these scores, which means which of these means I am closest to. The one that I am closest to, if it is, the distance to the closest mean, if it is large, then it means that I am away from all these means. So, if the closest guy is far away, then everybody else is far away.

So, I am going to give a score to each of the data points proposed as exactly the minimum distance to the means that I have seen so far. So let me put this formally, so this is like see, the score for point x is just the minimum, over all me- all j from 1 to l - 1, because I already have l - 1, means is the l-th round. What is the distance I am thinking of $(x - \mu^0_j)^2$.

So, I am computing the distance square to each of the data points each of the which I have chosen has mean so far, and then I am seeing which is the smallest distance, which is which point I make closest to. And that distance is what I am thinking of a score for a data point. Now, every data point gets a score, so for all x, this is for all x in the data set, I can compute the score.

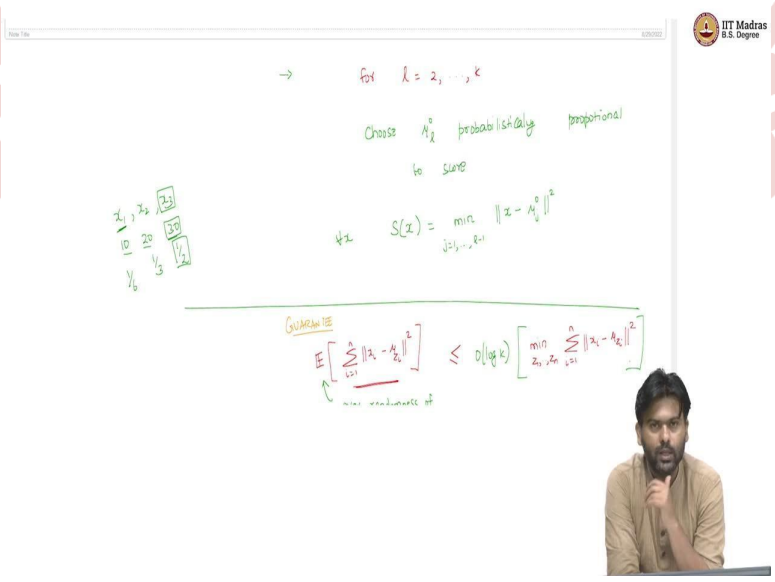
And now I will now sample the next mean, the next as a data point according to the score, so in a probabilistic fashion, which I explained earlier. So, now you are going to normalize these scores over all the data points and then you are going to probabilistically sample one data point like this. So, if x1, x2, x3 are the remaining data points, and x1 score is 10, x2 score is 20, x3 is 30, then x1, the probability that x1 gets chosen as 1 by 6, which is 10 by 60 and x2 gets chosen as 1 by 3, which is 20 by 60 and x3 gets chosen as 1 by 2, which is 30 by 60.

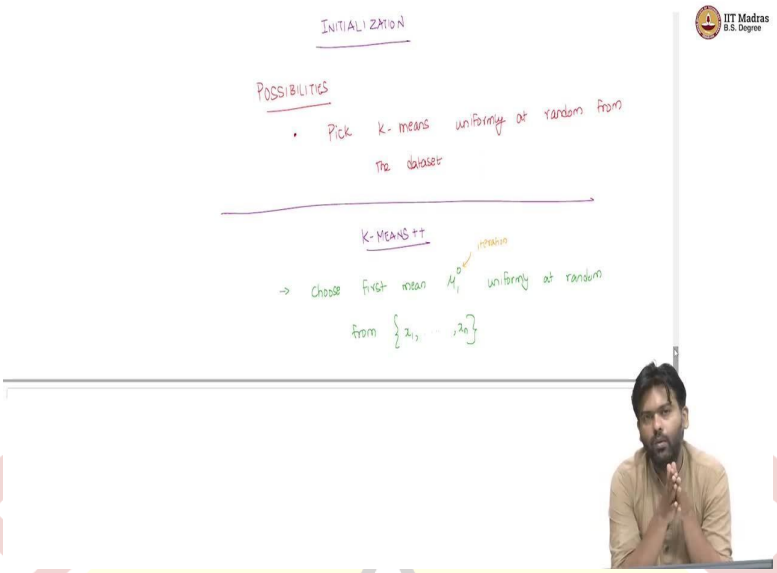
So, I want to sample according to x. So, this is the k-means++ way of doing initialization. One might wonder why am I doing this in a probabilistic way, why cannot I simply do this in a deterministic way? I want the means to be as away from things as possible. Why cannot they simply pick as of the next mean as the 1, whose minimum distance to the ones that have chosen so far is maximum?

I could have done that, so which means in this case, if there are 3 points, 10, 20 and 30, well I would pick x3 simply because x3 minimum distance to whatever that has been chosen, is much higher than the 2 which means it is much far apart than the means that I have seen so far. Well, what we have seen is that x3 still gets the highest probability of being chosen in k-means++, but it may not necessarily be chosen, it could be x2 also.

And you need this probabilistic way of doing this, instead of deterministic way, because you can show some kind of guarantee for this algorithm.

(Refer Slide Time: 13:45)





So, we I will brief as upon this guarantee, what this guarantee, what can you guarantee for this algorithm is as follows. And this to argue this guarantee, you cannot put down a deterministic algorithm, because once you put down a deterministic algorithm, you can come up with an adversarial data set where this guarantee might fake.

So, you want in some sense, an average guarantee and so you make the algorithm randomized. But then you do the most natural thing where you kind of introduced probability where the deterministic choice will get the highest probability, but then still other people might also get some probability so that on an average, you can make some kind of guarantees.

So, I will briefly touch upon the guarantee. So, the guarantee looks something like this. We will not go into the details or argue why this is true in this course, but then it is good to know that there is this underlying guarantee available. So, the expected value of $\sum_{i=1}^{n} (xi - \mu_{zi})^2$, which is the, this is just the objective of the partition that the algorithm results after a run k-means.

I do this initialization run Lloyd's, and then I end up with a partition and I can compute its objective value. Now, that objective values are random quantity, because my initialization was random. The Lloyd's algorithm itself is not random, so once you fix an initialization, everything else is deterministic in the Lloyd's algorithm.

But because the initialization is random, the answer final answer is also going to be random. So, which means I can ask over the randomness of the over randomness of algorithm, which is the initialization, how does this on an average how am I doing with respect to the objective function. What I really want or care about is the best possible thing, So, minimum $_{z1 \text{ to } zn} \sum_{i=1}^{n} (xi - \mu_{zi})^2$.

So, this is the best possible that I want, I want that partition, which gives me the least possible objective value, I may not be able to get it and that is an NP hard problem to get that. But what I am saying is here, k-means++ guarantees is that you may not be able to achieve that partition, which gets this but then on an average, you are not going to be too far away from that partition.

In some sense, you are going to be something like some constant times this quantity, something like this. Basically, what it is saying is that, let us say the best possible objective function gives me some value of 10. On an average, I am saying that the partition that k-means++ will result in is not going to be more than let us say, the objective value is not going to be more than let us say 5 times 10.

So, it is got to be more than it, but then it will not be too far away also, on an average, that is the kind of guarantee that you can give. The specifics of the guarantee, or how this comes about is beyond the scope of the course, we will not prove that. But it is good to know that, by doing reasonable initialization, you might end up in partitions, which are not too bad.

The downside of doing this is though, that you need to spend a lot of time doing this initialization, So, because if you had a billion points, and you want to choose 10 different means, now every time you need to compute these scores, and then you need to, do a probability, convert that to a probability sample, and so on, you can do some tests to make that faster, but then still, you need to go through this thing of doing this k different times.

So, computationally, this might be less efficient, or might take more time than just, our previous method, which is just pick k-means uniformly at random, and then try it out and try it multiple times and see what works. So that is faster, the uniformly at random method is faster, but then it is not, I would say principle. So it is not pushing your means in certain direction. But typically, in practice, especially when you have high dimensional data, people also use the uniform one extensively.

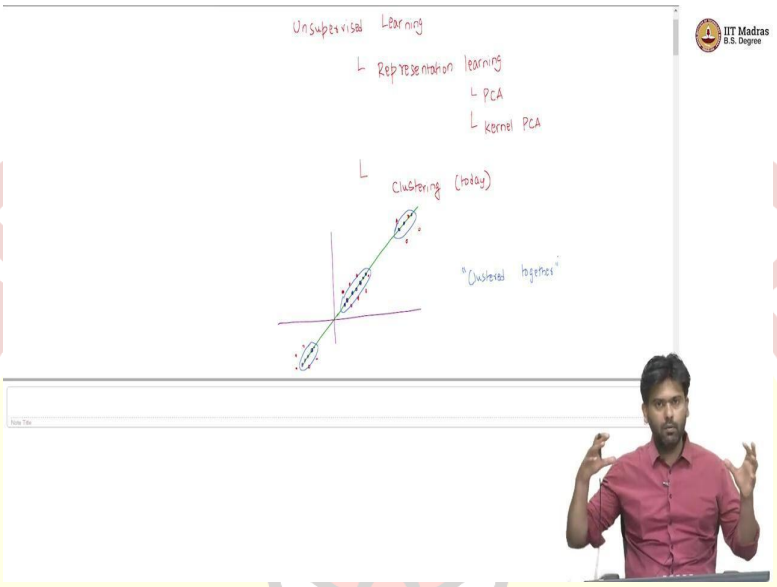
So now, it depends on how much time your algorithm, I mean, you have to run your algorithm, how much resources you have and all that will make you this, I mean, you can make a decision based on that. Nevertheless, k-means++ is a solid way to initialize your Lloyd's algorithm. So that is, what I wanted to comment about the initialization part. So, the final thing that we will talk about is the choice of k which we will see next.

IIT Madras

ONLINE DEGREE

Machine Learning Techniques
Professor Arun Raj Kumar
Department of Computer Science & Engineering
Indian Institute of Technology, Madras
Introduction to Clustering

(Refer Slide Time: 0:14)



Hello everyone, welcome back. So, far in this course, we have been looking at unsupervised learning. And specifically, we have been looking at representation learning problems in unsupervised learning, so representation learning. And in representation learning, we looked at the PCA algorithm, which is the principal component analysis algorithm, we also looked at a kernel version of it, which we call the kernel PCA.

What we are going to do now is look at another paradigm of unsupervised learning, which is called as clustering. So, this is what we are going to look at today. And this is also a popular unsupervised learning problem. So, what is the motivation for looking at clustering? Let us take a simple picture, two dimensional picture. And let us say we have a bunch of data points which are like this.

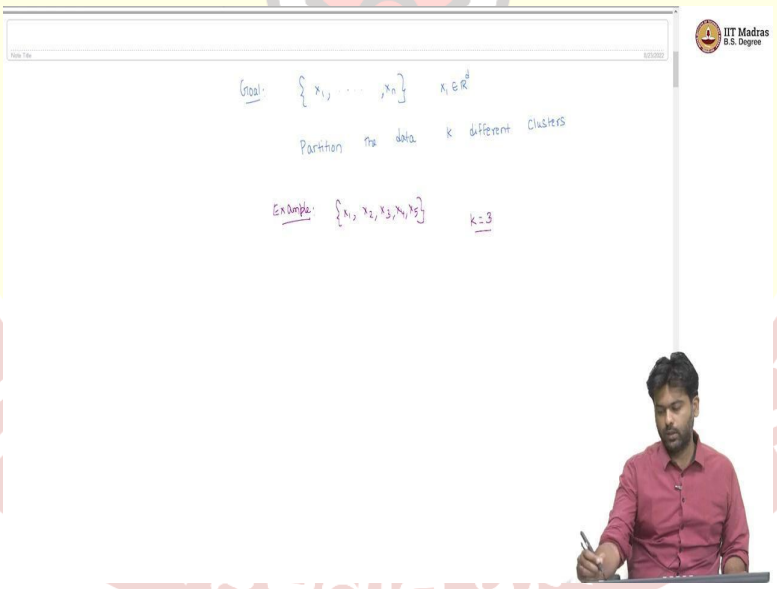
Let us say some points here, some points here and maybe some points here. Now, one thing we could do to represent this data in the way that we have seen so far is to do a PCA on this dataset. And if you do a PCA on this dataset, what we would get is the Eigen direction, which captures the maximum variance would be perhaps something around this direction along this direction. And once we have found the Eigen direction, what we typically do in PCA is find the proxies for these data points along the Eigen direction.

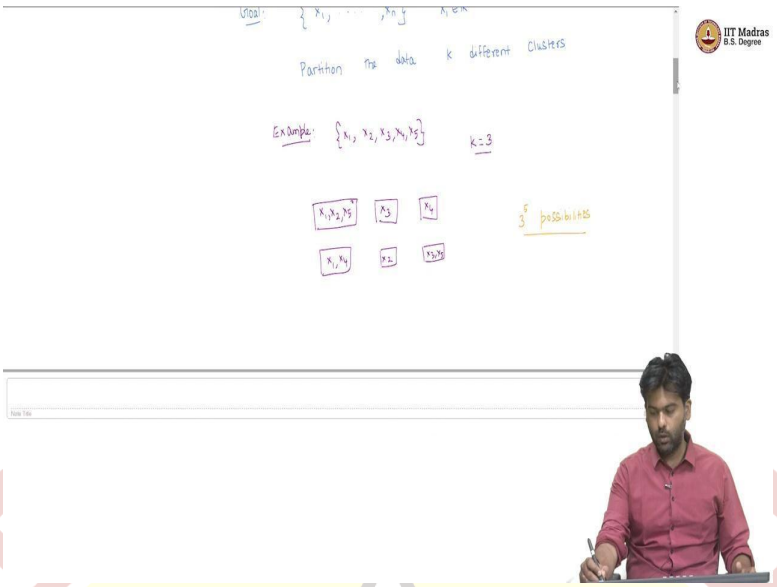
If we do that, then we are going to get points like this. I am not be exactly drawing it, but you get the idea. So, maybe there are more points here, here and then their corresponding proxies would be here, here. Now, the question is, have we really understood the underlying structure in the data? It is one thing to say that there is a line or a linear subspace that has most of the information that is present in the data.

But then in this case, that linear subspace happens to be this green line that I have drawn here. But there is more that you can say about this. So, it is not just the linear subspace, but then in the linear subspace, you still have some kind of information. So, the data points are what I would call as clustered together.

So, it is not that they are all over the place in the linear subspace that we are finding in PCA, let us say, but then even within the subspace, there is some more structure to be uncovered. The question that we lost today is, in general, if you are given a bunch of data points, how can you uncover these cluster based structure in the data.

(Refer Slide Time: 3:19)





So, the goal of the lecture is to understand let us say, we are given a bunch of data points as usual x1 to xn, all data points we are going to assume are in D dimension. Now, we want to partition the given data into K different clusters, K different partitions or clusters or groups. We can call them however we want. But that is the goal. So, we want to partition the data into K different, think of these as boxes, let us say.

So, here is an example. I mean, a simple example, let us say we just have a bunch of points, x1, x2, x3, x4, x5 and we want to partition them into, let us say, 3 boxes. Now, there are multiple ways you can group these things together. So, maybe there is one way which groups x1, x2, x5, in one box, x3 in one box and x4 in one box. Maybe this is a way to partition the data into 3 boxes or 3 clusters.

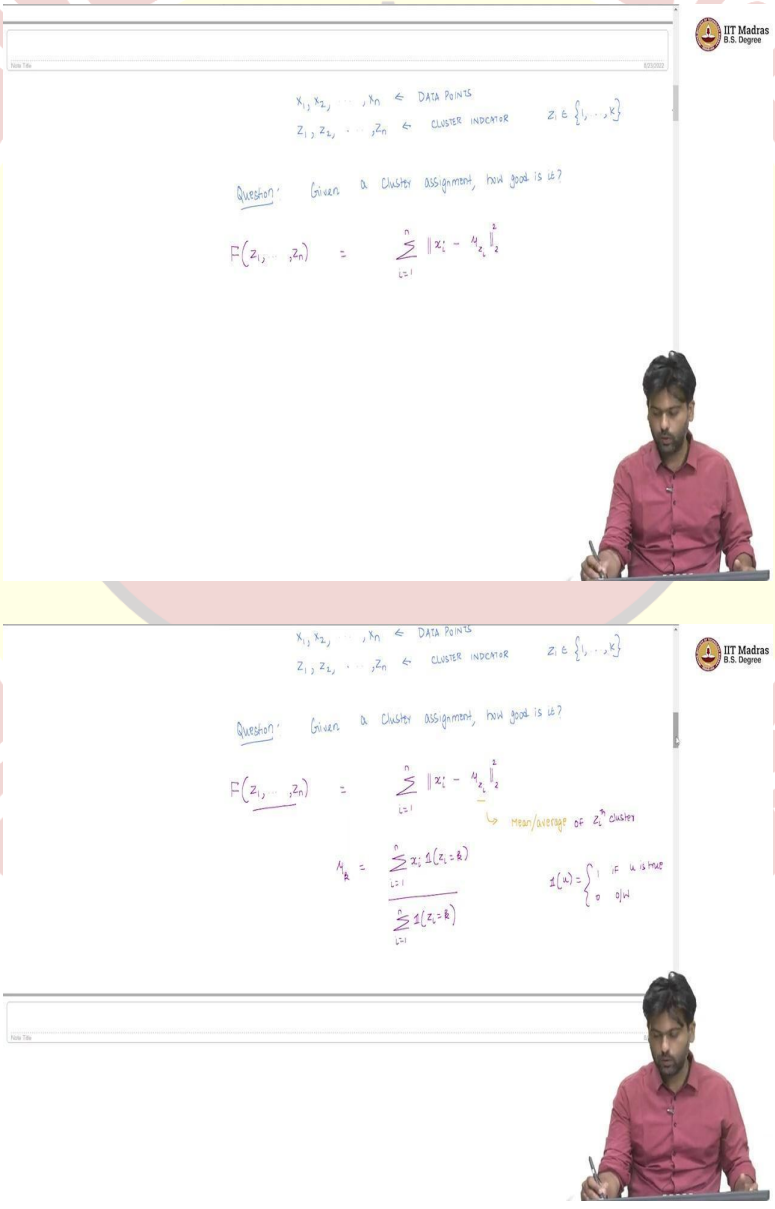
Here is another way, x1, x4 in this way, in this grouping group together, x2 stays separate, let us say x3, x5 group together. So, there are multiple ways you can partition a bunch of data points into boxes, K different boxes. In fact, in this case, if you want to argue how many ways are there to partition, 5 points into 3 boxes, naively speaking, each point, you take every point and then the point has 3 different options, 3 different boxes that it can go to.

And if you take the second point that also has three different options. So, if you naively compute how many ways you can put 5 points into 3 boxes, there are 3 into 3 into 3 into 3 into 3 in each point has 3 options. So, there are 3^5 possibilities. So, this also of course, includes possibilities which lead to empty boxes or empty clusters. But for the moment, even if you do not want that this will still be a very large number.

So, this is exponential in the parameters of interest, in this case, the number of data points and the number of points number of 2 boxes. So, this is something that we understand so that there are a lot of possibilities of putting points into a bunch of boxes. Now, we want to understand how do we, what is, what is a good way to do this.

That is the goal, which means that for each of these partitions, each of these ways of putting data points into boxes, we need to say, how good is that partition, we need to come up with a performance measure for a partition given a bunch of data points.

(Refer Slide Time: 6:14)



So, the next goal, what we want to do is what is a good performance measure for partitioning data points into clusters. Towards this, what we will do is we will first introduce some

notation. As usual, our data points are x1 to xn. So, these are data points. Now, we are going to say every data point is associated with a cluster indicator variable, which I am going to call as z. So, this is z1 to zn so these are cluster indicators, zi belongs to 1 to K.

Meaning every data point goes to one of these boxes, 1 to K. So, that is the problem. So, we need to put each point in one of the boxes, let us say if I put the tenth point in the fourth box, then x10 is the point data point and the corresponding z10, the variable corresponding to the cluster indicator for the tenth point which is z10, will now be 4.

So, now each of these values z1 to zn takes a value between 1 to K which indicates which cluster or which box, the corresponding point goes into, so this is a way to define a partition, if you will. So, once I give you z1 to zn then you know where each data point goes into. So, that is what this z1 to zn tells you and then that determines a partition. So, once we have this notation, then we ask the question given a cluster assignment, how good is it?

We want to understand how good a partition is? So, I can give you some cluster assignment and then I asked you how good is this assignment? Now, you need to objectively measure performance of goodness of a partition, which means that we need to associate a number to each partition, which means that if I tell you z1 to zn then you need to give me a number, which says how good is this way of partitioning x1 to xn via z1 to zn, how good is it?

So, let us call this a some function z1 to zn, of course, depends on x1 to xn also so I give you the partition partitioning of the data points into K clusters. I asked how good is this partition? So, now, you can one way to think about this, there are multiple ways you can define this function. One natural way you can ask is, you somehow want partitions to be homogeneous in the sense that if I say that I have clustered a bunch of data points into three different clusters, every cluster should somehow look alike.

Now, how can we measure alike looking alike of a cluster? Well, again, you can measure it in different ways. One natural way is to kind of look at how spread the data points are in this cluster. In other words, you can ask well, how different are these points from the mean or the center of this cluster.

So, you can measure the distance from of each point to its center and then sum it up over the set of points in the cluster and then that will give you some sense of how good is this cluster?

How homogeneous is this cluster, if the all the points are the same, then each point will suffer zero distance to the center. If the points are like, well apart, then they suffer a larger value.

So, we want to somehow formalize this intuition and say that the performance measure that we will be interested in is as follows. So, for every data point, you measure the distance of the data point xi to μ_{zi}. And I will tell you what μzi is and then this is an l2 squared distance. So, what is μ_{zi}, well, μ_{zi} is mean of mean meaning the average, mean or average of zi cluster.

Remember zi is a value between 1 to K, so it tells where which cluster xi goes to. So, how can we define μ's, μk can be defined as follows μk is just the average of all the data points which go to the kth cluster according to z1 to zn. So, I look at z1 to zn each of these is a value between 1 to K and look at which of the data points have been assigned to the kth cluster. And then I compute the average for that set of points.

So, in notation, one way to put this would be to do the following. I say I am going to sum up all data points. But then not all data points have been assigned to the kth cluster. So, I will multiply this with a number, which is an indicator of whether the corresponding cluster to which the xi point has been assigned to is k.

So, this indicator value takes 1 if this is true, which means if the ith point is been assigned to the kth cluster, then this 1, otherwise, this is 0. So, basically, what I am doing is I am adding up all the points which have been assigned to the kth cluster. And then I am dividing it by the number of points assigned to kth cluster, which can be just got by summing up the indicators over all data point.

This would again be 1 indicator of some u is 1, if u is true and 0 otherwise. Which means all I am doing is it is just a notation to say that I am just looking at each box and then computing the average of that box. So, then what does this performance measure compute, it computes the distance between every point to the mean of the box distance square of every point to the mean of the box in which it is assigned. So, this is the performance measure with which we will work right now.

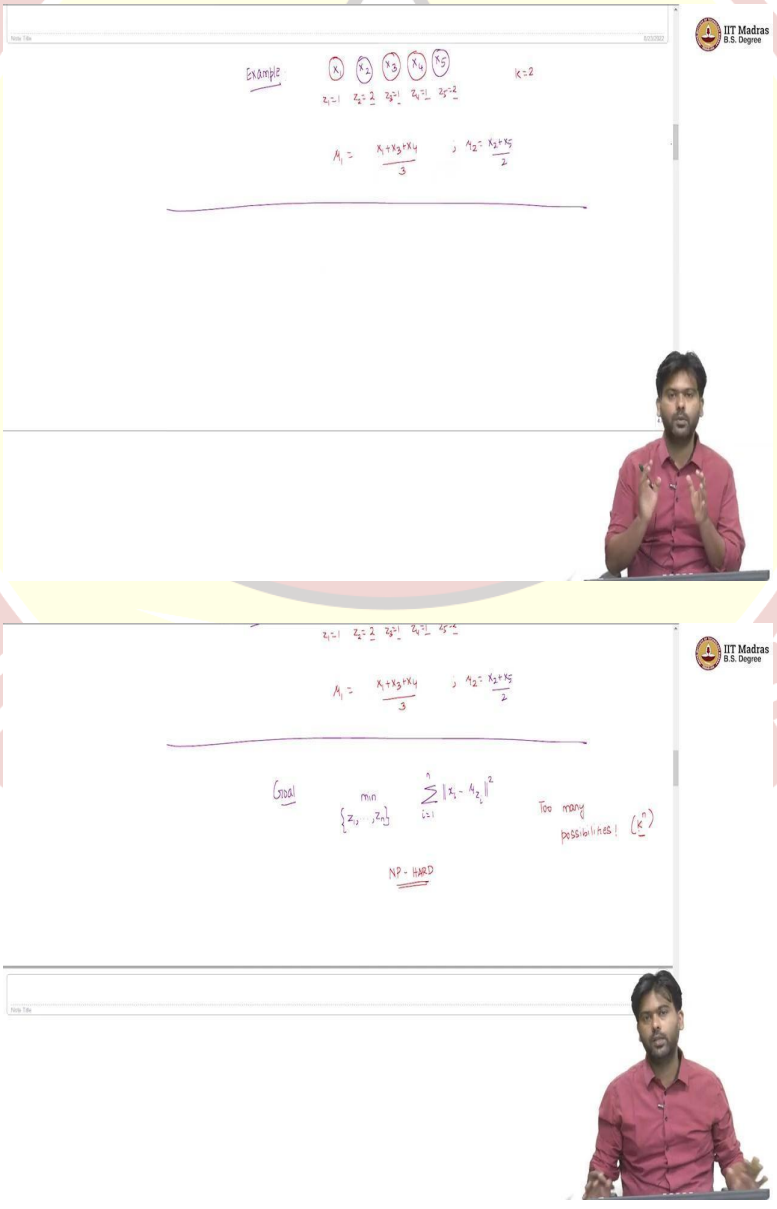
Just to make sure we all understand this perfectly well. So, here is an example. Let us say I have points x1, x2, x3, x4, x5 and then I have z1 is 1, z2 is 2, z3 is 1, z4 is 1, z5 is 2, in this case, let us say K is 2. I want to divide the 5 points into 2 boxes. And here is one way to

divide that, so I am saying x1 goes to the first box, x2 goes to the second box, x3 goes to the third box and so on, fourth to the one, fifth point to the second box.

So, what would be μ1? μ1 would be check which of the points have been assigned to box 1, in this case, x1 and x3 and x4, so this will be (x1 + x3 + x4) / 3, μ2 would be what is what are the points assigned to the second cluster x2 and x5 . So, this is going to be (x2 + x5) / 2.

So, this is basically an example of whatever we are saying here. So, the goodness of this partition itself would be $(x1 - \mu1)^2 + (x2 - \mu2)^2$ because 2 is the centers partition to $(x3 - \mu1)^2 + (x4 - \mu1)^2 + (x5 - \mu2)^2$ and so on. So that is the way we are defining partitions.

(Refer Slide Time: 14:04)



And so now we have put down a specific measure or a metric to measure goodness of a partition. So, what would be our precise goal, our goal is now to minimize over all possible partitions, the measure that we just put down, which is $\sum_{i=1}^{n} (xi - \mu_{zi})^2$. So, this is our goal.

So, when I want to find out this is to go over all possible partitions and then see, which one gives us the least value and we know that there are only a finite number of partitions. So, we could potentially imagine an algorithm where we will go over each partition and measure this and then pick the one that has the smallest value.

But what might be a problem with that algorithm. If you want to pause and think let me tell you the problem with that algorithm, now, the problem is there are too many partitions. There are too many possibilities to do this naive algorithm. In fact, the naive thing would be it will be K^n. So, each data point has K possibilities if there are K clusters and there are n points.

So, there are potentially K power and ways to divide put points into boxes. Again, this is an upper bound because this also takes into account empty boxes, but then still, your actual value will not be 2, it will also grow exponentially. And that is a problem. So, computer scientists would say that this is an NP HARD problem.

In other words, it is not expected that we are going to get an algorithm to solve this, whose run time the amount of time it takes to run is polynomial in the parameters of interest, which is n and k in this case, maybe if there was something like N^2 into K^3, that was your the amount of time if you came up with an algorithm, then that is a good algorithm because it runs polynomial in n polynomial in K.

But then right now we are saying the naive algorithm is going to take K^n, where you look at go over all possible partitions and that is a big no, no. So, we simply cannot run this algorithm. Imagine even if K is 2 and the n is if you have 1000 data points. And this is saying the number of possible ways grow something like 2^{1000}.

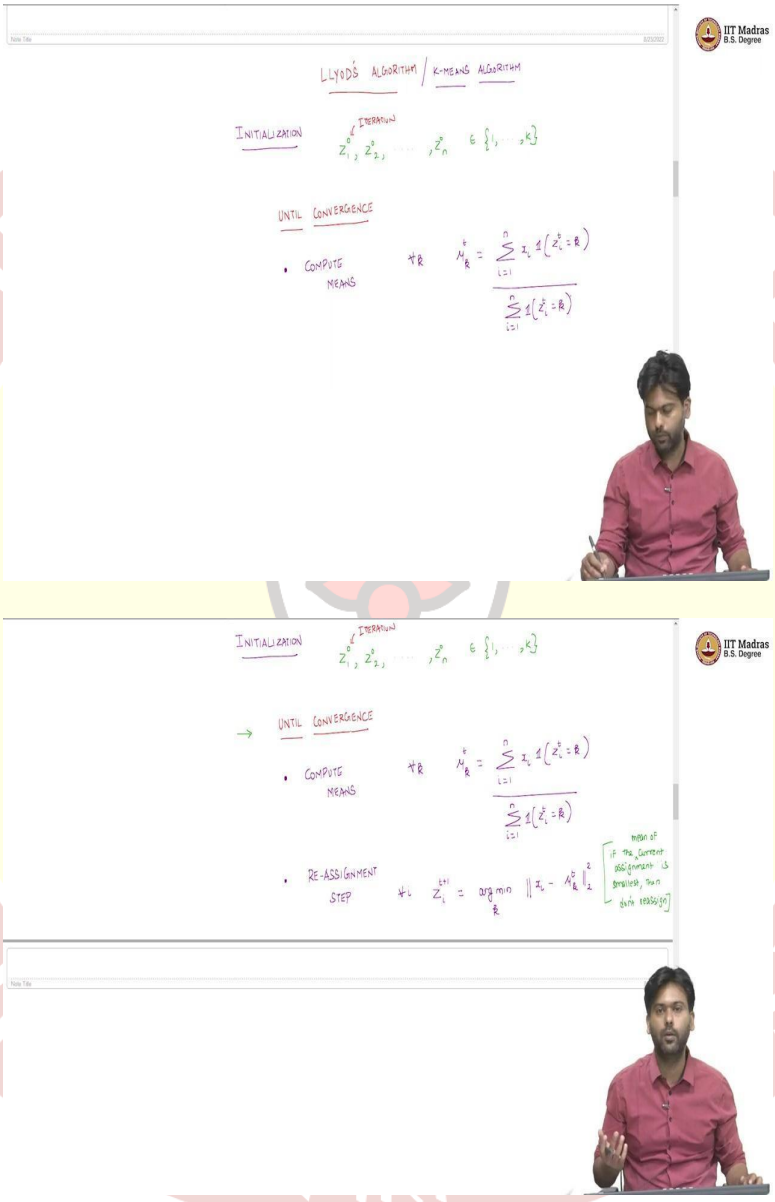
And that is like an astronomically large number to deal with. So, and we do not want to do that. So, then what can we do? So, if this is our goal, then how can we achieve this goal? We would not necessarily be able to achieve this goal exactly, but what we will do is we will come up with a very popular heuristic algorithm to kind of solve this problem. And we will see what this algorithm next.

IIT Madras

ONLINE DEGREE

Machine Learning Techniques
Professor Arun Raj Kumar
Department of Computer Science and Engineering
Indian Institute of Technology, Madras
K-Means Clustering (Lloyd's algorithm)

(Refer Slide Time: 0:14)



The algorithm that we are going to see is called as the Lloyd's algorithm or sometimes also referred to as the K-means algorithm. Though this is a misnomer the problem is the K-means problem you are trying to find K-means, which represents the partitions. But sometimes the algorithm is also called as a K-means algorithm though it is the K-means problem for which the Lloyd's algorithms a solution is a possible way to solve this problem may not be exactly but then a heuristic way to solve this problem. Anyway, so names aside. So, let us look at

what this algorithm is right now. So, it is very, very simple algorithm. So, it does the following.

So, the first step is initialization. We have a bunch of data points, which we need to first put in some boxes. So, we will talk about what are good ways to initialize a little later. But for now, for the moment, assume that we are starting with some way to put points in boxes, which means that we have z^0_1, z^0_2 till z^0_n everything in between 1 to k. So, this 0 here indicates the iteration number. So, at initialization, you are saying each point has been assigned a cluster indicator, some value that you can get started with.

Now, what is the algorithm do? The algorithm essentially does two simple things. It runs until convergence. And we will talk about what it means to say the algorithm converges in a bit. But let me put down the algorithm first. The first step is compute the means step. This is step 1, which means what we do is for all k, we compute μ_k, which is the mean in the t th iteration as simply the way we defined earlier $\sum_{i=1}^{n} x_i \, 1(z_t^i = k)$. In the t th iteration does your partition look like put i th data point in the kth cluster divided by $\sum_{i=1}^{n} 1(z_t^i = k)$.

Well, every round, you are creating a new partition. That is the idea. And what we are saying is that once the partition is created points go into boxes and then you can compute the mean of each of these boxes. And these means we are just going to call them μ^t_k to indicate that we are in the t th iteration. That is the easy point.

So, the main crux of the algorithm relies on this step that I will put down now it is called the reassignment step, which says how am I going to reassign the partition. So, which means that I start with some z's, which is z^0 now in the next round, I need to change it to an updated cluster indicator values. How do I do that, so for all i for every data point, I need to say which cluster it goes to now, in the next step, so I am going to do the following. I am going to do reassign z^i_{t+1} which is the cluster indicator for the ith data point in the next round as the following.

So, basically, what I am doing is, I have a partition which means points are being put inside boxes every box has a mean now, I look at every point and then compare its distance to the mean to the box in which it is assigned to, to the mean of every other box. If I find a box, whose mean is strictly less than the distance to the mean in a different box is strictly less than

the distance to the current mean inside the same box. Then I assign this data point in the next round to that other box which means which is exactly what is happening here. I am saying z^i_{t+1} the assignment to the i th data point in the t plus first iteration is just that k that minimizes the distance of the point to the mean of each of these boxes, which box has a mean whose distance I am who whom I am closest to.

So, this is assuming if the current assignment mean, so if the mean of current assignment is smallest, then don't reassign. So, maybe you are in a situation where the current points, the current means distance is exactly the same as the distance or the mean to a different cluster. In that case, I do not want to jump, I will still be in the same box where I am. But if I find a box whose mean is strictly less than the current distance, so the current mean, then distance square to the current mean, then I will make this jump. So, that is what is the understanding. So, this is all the algorithm is.

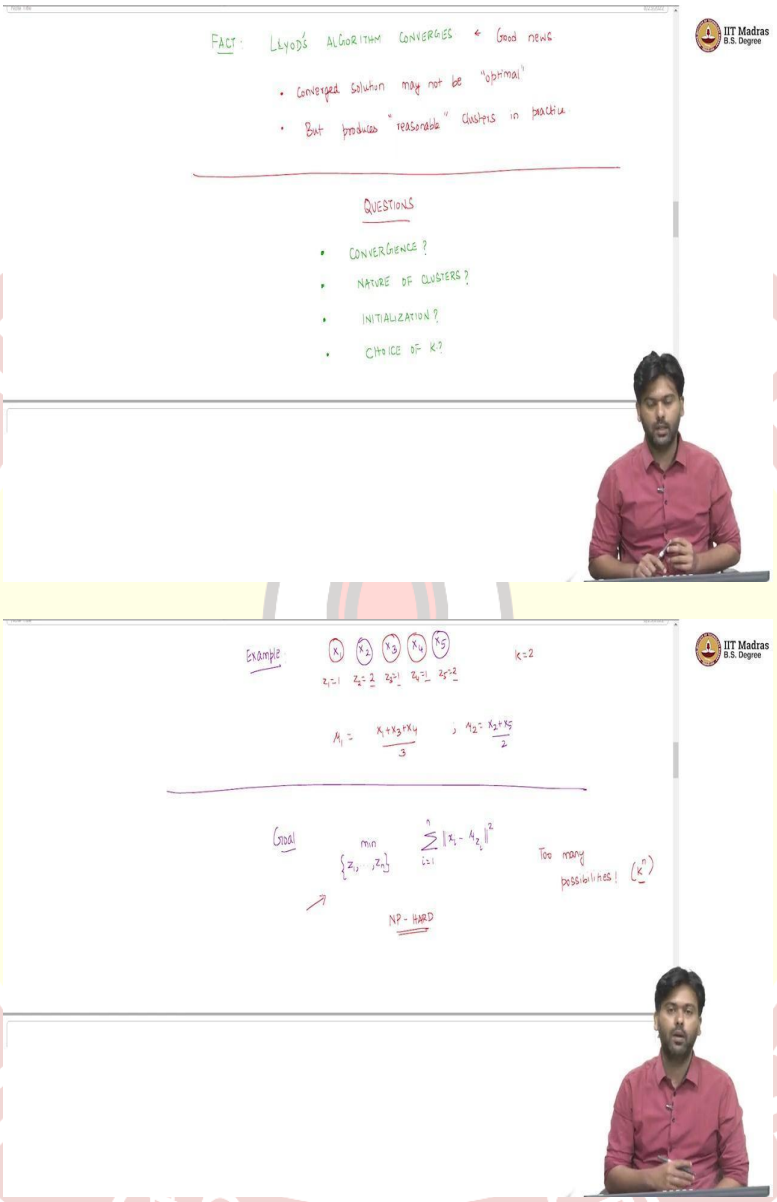
Now we need to talk about a few things here. So, what does it mean to say, this algorithm converges? Well, what is happening in each round is that I am changing the partition from one to another. So, at some point, if I encounter a situation where no point wants to jump, partitions, jump boxes, so every point is happy with its own box, which means that the distance of each point to its own box's mean is strictly less than any other box's mean, then no jumping around happens and then the algorithm is said to have converged.

The question is, I have only put down what until convergence, which means that we need to first see if this algorithm converges at all, so it could happen that I am in a certain partition and then I change in round two to a different partition and then round three to a different partition, and the next round, I come back to the original partition, if that happens, I will be stuck in a loop. And then I will keep moving around and I will never change from one partition to another. I mean, I will never converge, the algorithm will never converge. So, I will keep changing partitions, but then I will never converge.

But then here, I am saying until convergence, which means that we need to first understand if this algorithm converges at all, so all this algorithm is saying that hoping that this is that I am in some partition and I hope that I go to a better partition after making this change. So, because intuitively, we want partitions to be to all look similar. So, every point should be closer to its own mean than any other mean. If that does not happen, then we make a jump. And we hope that this will eventually lead to a partition, which is a reasonably good partition.

But then that is a hope at this point, we need to argue why this algorithm will converge and so, I mean, will it converge, if so, how do we argue that? So, that needs an argument.

(Refer Slide Time: 8:31)



The fact is that Lloyd's algorithm converges. And we will see why a little later but we can argue that this algorithm indeed converges. That is the good news. However, it could also happen that the algorithm does not necessarily converge to the optimal solution. So, the converged solution may not be optimal. Meaning it may not be necessarily the solution to the original problem that we put down, which was to, which was this np hard problem. Obviously, we do not expect to solve this original problem because it is known to be hard. So, whatever the algorithm results in may not necessarily solve this problem. So, that is the that is

something that we have to live with in some sense, but produces reasonable clusters in practice.

So in practice, this is a very popular algorithm and it kind of produces not so bad clusters. So, what we will do now is ask a couple of questions about this algorithm and then try to answer each of these questions. So, now I put down some algorithm. So, there are lots of questions that one needs to ask about, what kind of datasets where this algorithm will work well and there are many other questions. We will first list on some of the important key questions about this algorithm and we will try to answer those questions in the as we go long. So, here are some questions.

First question is about convergence, does is algorithm converge. The second question is, so let us say it converges, what kind of clusters does it produce, the nature of clusters. The third question on my task is, so there are some things which have not been specified clearly in the algorithm, one of the things is initialization. How do I initialize this algorithm?

And finally, so I have kind of side stepped this issue that we are assuming so far that given a set of points, we know the number of boxes that the points can have to be clustered into. But nobody is telling us that we are just given a bunch of points and then in an unsupervised way, we need to figure out some natural groupings, which means that what is the k that we should use that works well for this data set.

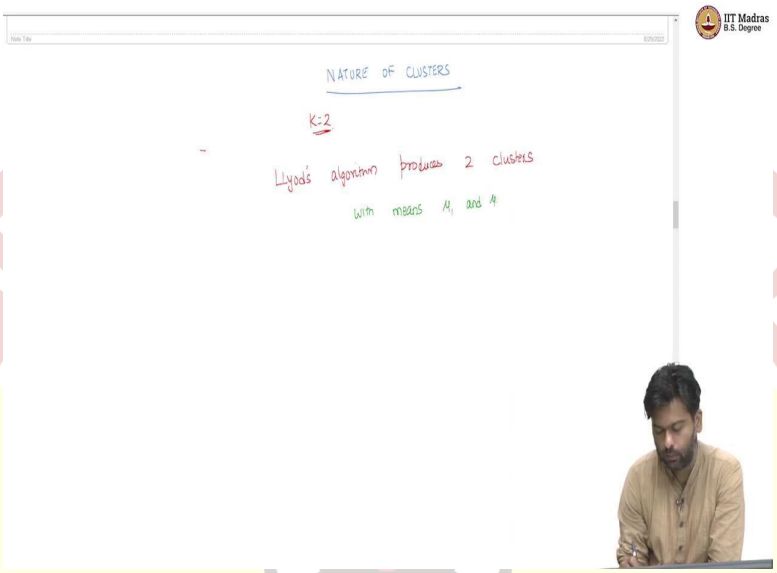
So, the choice of k is also question that one needs to ask. So, we will try to answer all these questions in the following part of this of this general discussion about the Lloyds and the K-means algorithm.

IIT Madras

ONLINE DEGREE

Machine Learning Techniques
Professor Arun Rajmar
Department of Computer Science & Engineering
Indian Institute of Technology, Madras
Nature of Clusters Produced By K - Means

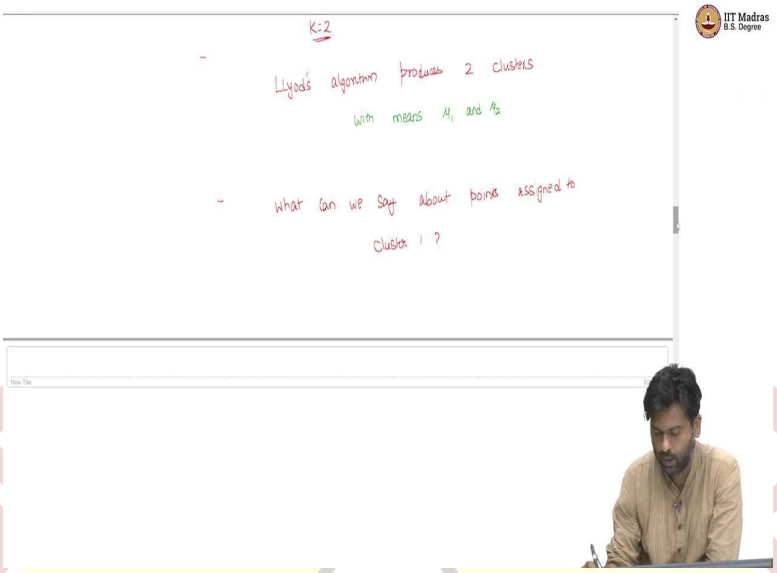
(Refer Slide Time: 0:15)



So, the next thing we are going to look at is what kind of clusters are being produced by the Lloyd's algorithm? So we know that the algorithm converges, but what can we comment on the type of clusters that that results from this algorithm?

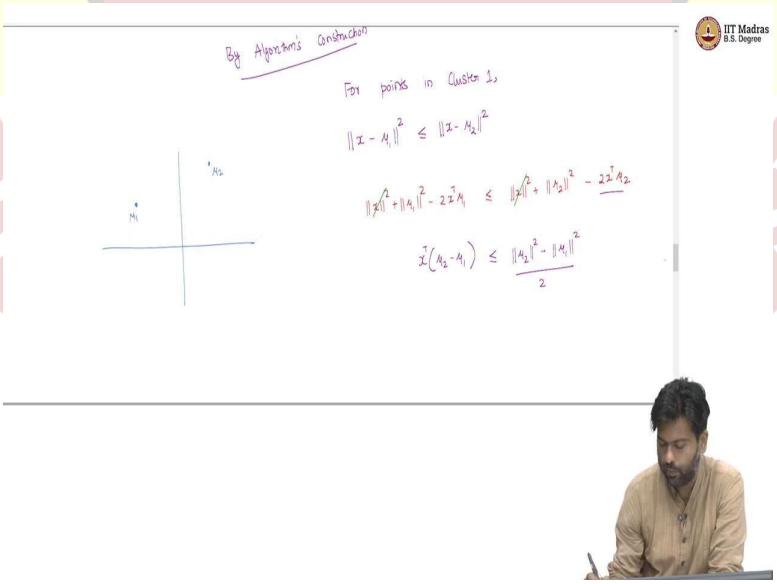
So to take a simple case, let us look at the case first, when K equals 2, when you have a bunch of data points, you just want to divide them into two buckets or two clusters. So let us say Lloyd's algorithm is run with just K equals two and Lloyd's algorithm produces 2 clusters. It will converge and it will produce two clusters with means, let us say μ_1 and μ_2.

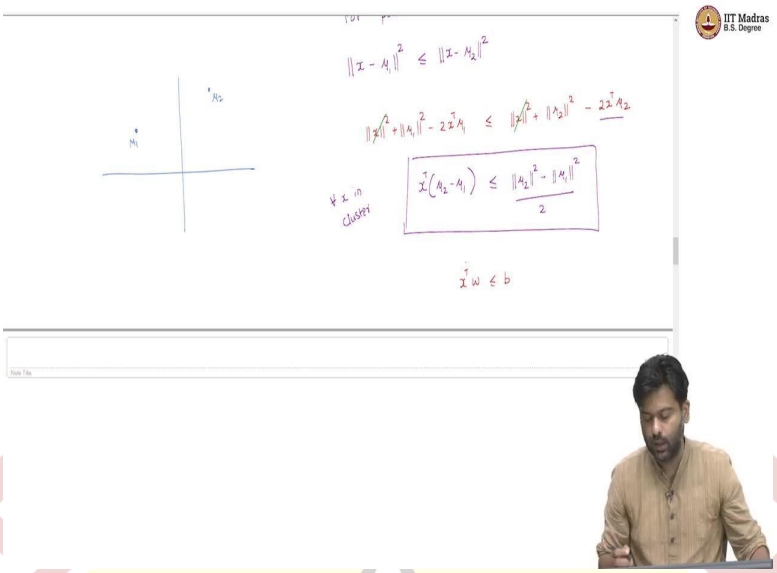
(Refer Slide Time: 1:30)



Let us say the algorithm converges and then we, in the converged partition clusters, we look at the average of cluster 1 and cluster 2, and it turns out to be μ_1 and μ_2. Now, the question is, what can we say about the points assigned to these clusters assigned to cluster 1 and cluster 2? Let us say cluster 1. Let us say we focus on cluster 1, the argument for cluster 2 will be similar.

(Refer Slide Time: 2:10)





Can we say something about how do the points look like? Well, by algorithm's construction and the convergence argument that we did earlier, we know that when the algorithm said, the partitions have converged, it means that everybody, every data point is happy with their own mean. What does that mean?

Well, for if you look at cluster 1, then it means that each data point that has been assigned to cluster 1 is closer to μ_1 than μ_2 in terms of distance squared that is what it means, right? So it means that for cluster 1, for points in cluster 1, $\|x - \mu_1\|^2 \leq \|x - \mu_2\|^2$. Where x is of course, a point a generic point in cluster one, right?

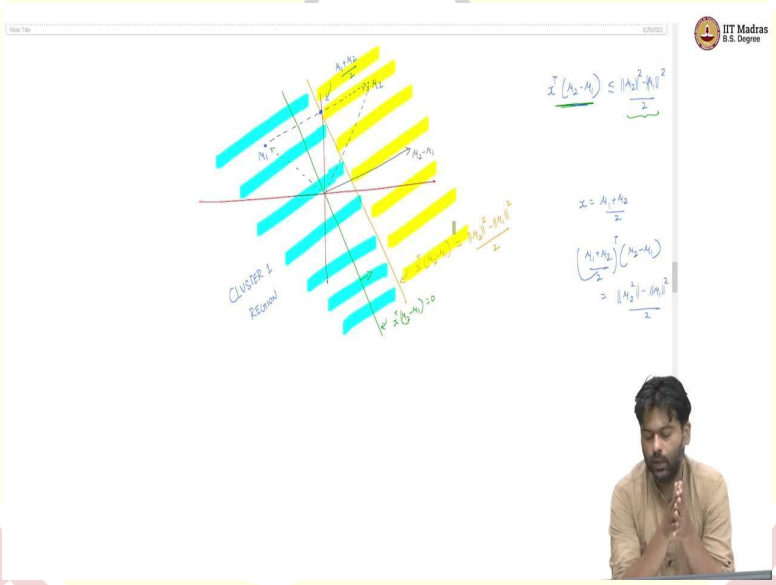
So its distance squared to μ_1 should be at most, the distance square to μ_2. Now, what does this mean? I mean, how do we understand this? So we can think of this as μ_1, this as a μ_2, let us say I just give you μ_1 and μ_2. And they ask where could be these clusters? Now, we know that points assigned to cluster 1 satisfy this equation.

But what does this equation, let us unravel this equation a little bit and see what happens? So, this is just $\|x\|^2 + \|\mu_1\|^2 - 2x^T\mu_1 \leq \|x\|^2 + \|\mu_2\|^2 - 2x^T\mu_2$, of course, we can cancel certain terms and what would result is the following. I can bring the last term to the other side.

So that would mean $x^T(\mu_2 - \mu_1) \leq \frac{\|\mu_2\|^2 - \|\mu_1\|^2}{2}$, what is this? So this is for all x in cluster 1, it must be the case that this is satisfied. Simply because the distance of this points to μ_1 is less than μ_2, what does this mean? So pictorially what are these points? That is something that we want to understand.

So, if this is μ_1 and μ_2, how can we say where are the set of points which are, which satisfy this equation, this as you can already observe, this is an equation of the form, you know, $x^T \leq b$. So it is, it is, it is linear in x that is what we observed. So, which means we should expect some kind of linear division and let us see what that is?

(Refer Slide Time: 5:19)



So here is a picture we have μ_1, let us say this is μ_2. And we want the equation is $x^T(\mu_2 - \mu_1) \leq \frac{\|\mu_2\|^2 - \|\mu_1\|^2}{2}$. Well, basically the dot product of your cluster point x with respect to μ_2 minus μ_1 is at most something that is what this is saying. But let us look at where is the vector μ_2 minus μ_1, well here is the vector μ_2.

So, this is μ_2, this is μ_1 where is in this case? Let me just confirm the order, one order here, sorry, yeah, this is correct. So where would be $\mu_2 - \mu_1$. So, $\mu_2 - \mu_1$ is what you would add to

μ_1 to get to μ_2, which means it is this. So, this is what you would add to μ_1 to get to μ_2 that is your $\mu_2 - \mu_1$. So, which means the actual vector is from the origin pointing in this direction.

So, that is my $\mu_2 - \mu_1$ vector, this is where the vector that vector is. Now, we know where are the set of points which make us 0 dot product with this vector. Well, that would be somewhere here. So, this would be set of all x such that $x^T(\mu_2 - \mu_1) = 0$. But our condition says that $x^T(\mu_2 - \mu_1)$ is at most $\frac{\|\mu_2\|^2 - \|\mu_1\|^2}{2}$..

So, now, whether that value is positive or negative will depend on which length is bigger $\|\mu_2\|^2$ length is bigger or $\|\mu_1\|^2$ length is bigger, at least in this picture, it looks like $\|\mu_2\|^2$ length is larger than $\|\mu_1\|^2$ square length so, this seems to be a positive quantity. So which means if I want extra, if I asked the question, where is $x^T(\mu_2 - \mu_1)$ is less than or equal to instead of less than or equal to if I put the equal to and ask where is it equal to $\frac{\|\mu_2\|^2 - \|\mu_1\|^2}{2}$.

In other words, if I ask whether the set of all x which satisfy$x^T(\mu_2 - \mu_1) = \frac{\|\mu_2\|^2 - \|\mu_1\|^2}{2}$, well, because we are thinking of this as a positive quantity, that means that I have to move the green line parallel in this direction. So, that would give me a line let us say like somewhere like this.

Now, this could be the line which is $x^T(\mu_2 - \mu_1) = \frac{\|\mu_2\|^2 - \|\mu_1\|^2}{2}$. So then, where is our cluster 1? It says that any x that is in cluster 1 should satisfy something, is it has to be less than or equal to this, which means that it has to be in this region. So left of the orange line. This is the region for cluster 1, cluster 1 region. Now, this is one way to think of this, directly from the picture.

Now we can also ask, well, how much did they move on to the right? So I did not move the green line parallely to the right. So, how much did I move? Well, can we find a point which satisfies this equation, then we will know. So, one point that satisfies this equation is like take $x = \frac{\mu_1 + \mu_2}{2}$ which is the average x, so average of the means.

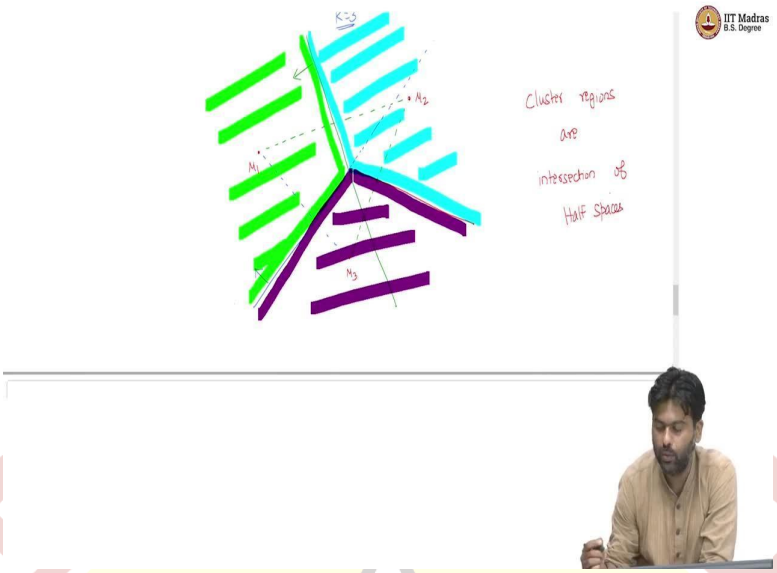
So now then $\left(\frac{\mu_1+\mu_2}{2}\right)^T(\mu_2-\mu_1)$, you can verify will equal $\frac{\|\mu_2\|^2-\|\mu_1\|^2}{2}$, which means that the average of the means is actually here. So the average of the means is exactly this point. This point is essentially then saying this is $\frac{\mu_1+\mu_2}{2}$.

So this is just $\frac{\|\mu_2\|^2-\|\mu_1\|^2}{2}$, which means this point, which is the average of, which is between μ_1 and μ_2 actually falls on this line. So, which is in fact, this is $\frac{\mu_1+\mu_2}{2}$. So, another way to think about this is to simply say that, well, I have two points. And then I want to, I want to understand the set of all points which are closer to one point than the other

Well I connect the line between these two points, look at the perpendicular bisector of this line, and then say that every point that falls on one side of the perpendicular bisector or closer to that point, and which is exactly what we kind of argued, in the general sense in high dimension as well.

So this is good so this just means, what happens in two dimension? So even K equals 2, let me just make sure that, well, if this is cluster 1, well, where are cluster 2 data points, cluster 2 on this side. So, basically what has happened is that, so your K means algorithm or the Lloyd's algorithm is going to give you clusters, if K equals 2, you just separate it by a line that is what this means. So it has to happen. This simply comes from the convergence criteria for algorithm, so that implies this.

(Refer Slide Time: 11:38)



So now what happens when K equals 3? That is an actual question we can ask. And we will see that and that might give us some more intuition as to how these clusters look like. In $K = 3$, the algorithm is going to converge with not just 1 mean, but then 3 means sorry, not just 2 means but then 3 means.

Let us call these means μ_1, μ_2, and μ_3, let us say again, the data is just two dimensional data, I mean, all of whatever we are saying works for high dimensional data also, but then it is easy to visualize the two dimensional data that is why we are doing this. So now, what is the question we are asking, where are the points which are assigned to cluster 1? That is the first thing that we like to understand.

Now, we know that by the convergence criteria of Lloyd's, every point that is assigned to cluster 1 is closer to μ_1 than μ_2 or μ_3. So, both of them, I mean, if I compare even if I find one of the others, μ_2, μ_3 as being closer, then I would have jumped the algorithm would not have converged. So that cannot be these μs cannot be the final μs.

Because the algorithm has converged, every point assigned to cluster 1 necessarily should have distances to μ_1 closer than both μ_2 and μ_3. So how does that look in picture? Well, first, where are the set of points which are closer to μ_1 than μ_2? Well, we already argued that that is going to be simply, let us say, if I put this imaginary line that joins μ_1 and μ_2, then that is just going to be in the left hand side of the perpendicular bisector of μ_1 and μ_2.

Now, μ_3 need not be on this line, μ_3 could be somewhere here also, right? So this line is like this. But we are also saying just for consistency, I am going to use the green color here. So, anybody on the left hand side of the green line is closer to μ_1 than μ_2, but that is not enough to say that point can be assigned to cluster 1.

So, it is also necessary that the point should be closer to μ_1 than μ_3 also, which means that I need to also look at the perpendicular bisector of the line joining μ_1 and μ_3 and that may be somewhere like this and it can go further, further is not that necessary. But now it means that well, the points assigned to μ_1 or on the left hand side or the direction that I am drawing here with respect to μ_3 and should also be on the left hand side that is the direction I am drawing here with respect to μ_1.

Which means if I had to shade that region where the points are in cluster 1, it will be an intersection of these two regions, which will exactly be this so, it will be the boundary would be something like this. Now what about μ_2 and μ_3. So where are the points in cluster 2 and cluster 3? Well, for that we need to draw one more perpendicular bisector, which connects μ_2 and μ_3 and that would some be, somewhere like this.

And that would tell us those assigned to cluster 2 are going to be in this region. So this would be my boundary, it is closer to both, I mean, it is closer to μ_2 than both μ_1 and μ_3 so, it should be in the blue region. Naturally, whatever remains is the ones that are for μ_3 region which is this region.

So, as you can see, what has happened is that you know, you have three different separators one for each pair of means, μ_1 μ_2 separator, μ_1 μ_3 separator and μ_2 μ_3 separator and then the points assigned to μ_1 or closer to μ_1 than μ_2 via the μ_1 μ_2 separator and μ_1 than μ_3 via μ_1 μ_3 separator.

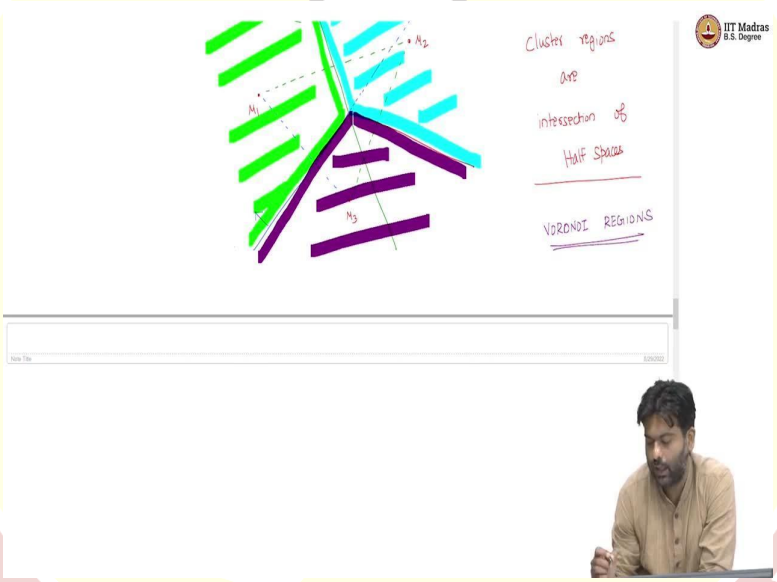
So, essentially what we are doing is that, we are dividing the entire space in this case R^2 which is where the data points are into K different regions using you know intersection of half spaces. Basically, when I say half space, it means that in two dimension you draw a line and anything on one side of the line is what is called as a half space, it is dividing the entire

space into two parts, a half is on one side of the line the other half is on the other side of the line.

So there are for every data point, so for every cluster, now, your region would be an intersection of $K - 1$ half spaces, if you have K clusters to begin with, because every point is going to get compared with, every mean is going to get compared with $K - 1$ other means and then it should attract point towards itself.

So it will be an intersection of half spaces. So cluster regions are intersection of half spaces. So this argument works even for high dimension with visualizing in 2d, but then it is true for high dimension as well.

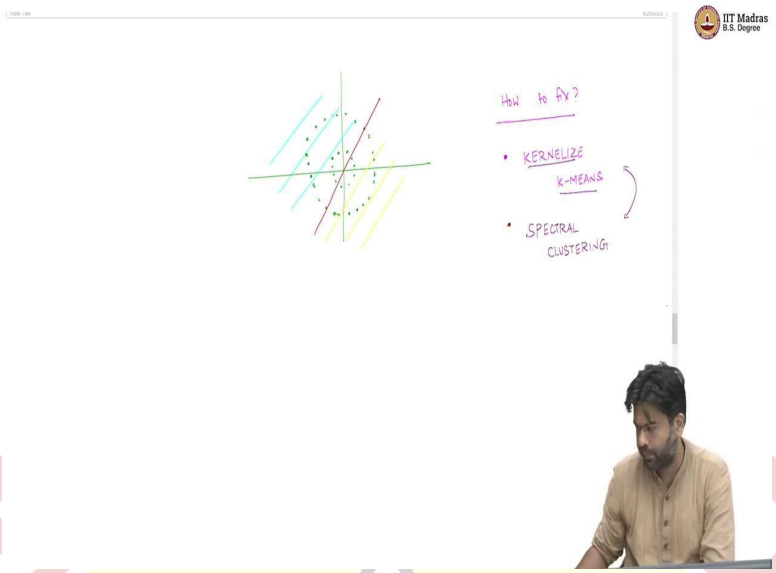
(Refer Slide Time: 17:20)



And these type of intersection of half spaces they also have a name, this is called as Voronoi region. So basically, what that then tells us is that if you run the Lloyd's algorithm then the clusters can be, you know, imagined as if they are falling in different Voronoi regions. Basically, your Lloyd's algorithm is trying to find the best, in some sense, Voronoi partition that will lead to, in some sense, the best cluster, right?

So you necessarily, the partition, the points assigned to each cluster should be part of a Voronoi region. That is what this argument finally says this. So this is the comment that I wanted to make about the nature of clusters.

(Refer Slide Time: 18:25)



Let me also, at this point, make a comment about something that we perhaps will not discuss in this course. But then it is interesting to know also. So for example, if I had data in 2 dimension like this, so let us say I had data around a circle, like this, and I had some more data points in a smaller circle.

Now, the number of clusters in this dataset naturally looks like two clusters and one cluster, which are in the outer concentric circle, and the other cluster are the points in the inner concentric circle. Now let us say I give you this dataset, and I asked you to run the Lloyd's algorithm with $K = 2$. Of course, it is going to converge, we know that, but what kind of partitions would it result in?

In other words, where would these points get clustered? How would these clusters look like? Now if you think about that, we know from the previous argument that the clusters necessarily have to be in Voronoi regions, if $K = 2$, then it means that there is a line that divides cluster 1 from cluster 2.

Now, if the points are in the circles, maybe you might get a line something like this. It is not necessarily that it has to be a slant line, it could be a vertical line, I mean that the exact line depends on the exact spacing of these points and so on. But irrespective of how the points are spaced, it is going to be a line which means that I am going to say anybody on this side is in cluster 1 and all the points are on the other side or in cluster 2.

Which means we did not recover the real clustering that we want, which was the inner concentric circles should become one cluster, the outer concentric circle should become

another cluster, your K-means algorithm in the way that we have described, it will never be able to recover such a partition, because the nature of clusters or Voronoi partition.

So, if your data set, if you do not believe that if your data set has a nice Voronoi like structure, where the partitions, where the cluster sit, then perhaps your K-means algorithm is going to fail there. So it will not be able to recover. Now, how to fix this, so how to fix? So, we will not do this in detail in this course, but let me at least give you some pointers, which you can perhaps if you are interested, you can go ahead and read about these things.

So, like how we argued in PCA that, if your features are non-linearly related, then you can use this notion of kernels to go from low dimension to high dimension. And in the high dimension, you learn a linear relation, which translates to nonlinear relationships in the lower dimension.

Similarly, you can, there is a way to, you know, kernelize K - means it is called the Kernel K means algorithm and it has interesting relationships to other types of popular clustering algorithms also, which we are not seeing in this course, for example, something called a spectral cluster which is, which has a lot of relation to PCA as well. So, in simple terms, you can understand this as if, you know, maybe there are no Voronoi regions in the low dimensional space, that gives me natural clustering of this data.

But then if I map it to some high dimensional space, then it would, it would kind of, you know, the natural clusters would kind of separate out into different Voronoi regions. And in the high dimensional space, I can do a simple Lloyd's algorithm that is the basic idea. So to go to this high dimensional space without really, the problem of computational efficiency and so on, inefficiency, and so on, we use the kernel trick, and you can Kernelize K- means also.

Let me leave it at that without dwelling into depth about how to do this Kernelization. But it is good to know that you can kernelize K means also. And this kernel version of K means has a lot of relation to, you know, computing the eigenvectors of your Kernel matrix and then using them somehow as some representation of these data points to do further cluster. We will not look at that in this course, but it is good to know.

So that is all I want to comment about the nature of clusters that Lloyd's algorithm produces. We have two more issues that we need to talk about. One is the initialization issue. What,

how do you initialize Lloyd's algorithm? Well, and the second is the issue of how do you choose the number of clusters? First, we will talk about the initialization issue.