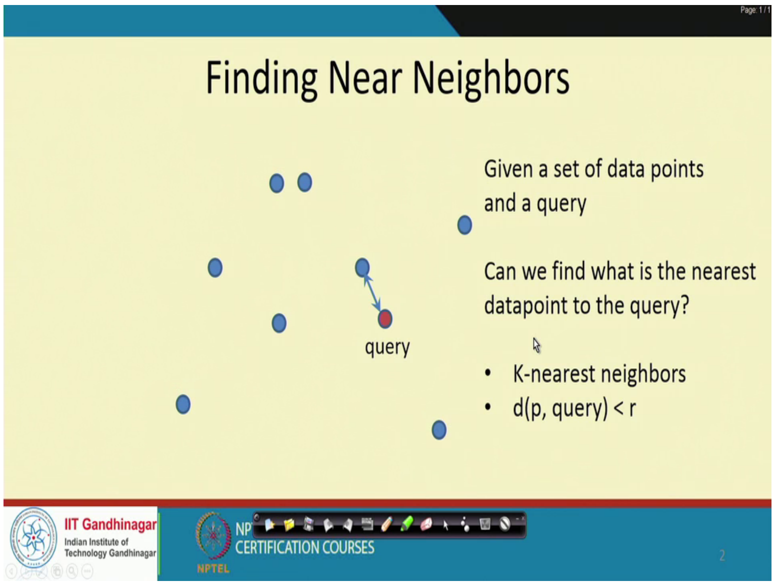
Scalable Data Science

Prof. Anirban Dasgupta
Department of Computer Science and Engineering
Indian Institute of Technology, Gandhinagar

Lecture - 11
Near Neighbors

Welcome to today's lecture of Scalable Data Science. Today, we going to start a slightly new topic. And, will start asking a question of how to answer near neighbor queries ok. I am Anirban and I am a faculty of Computer Science and Engineering at IIT Gandhinagar.

(Refer Slide Time: 00:38)



Finding Near Neighbors

Given a set of data points and a query

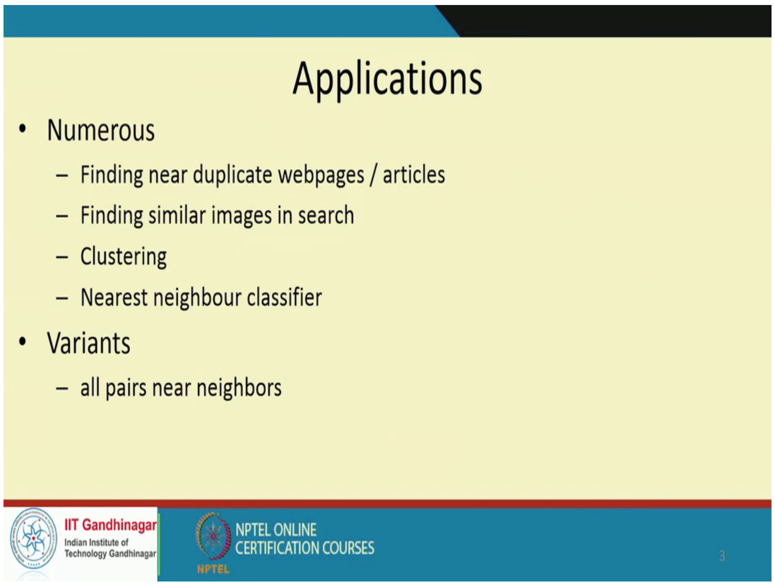
Can we find what is the nearest datapoint to the query?

- K-nearest neighbors
- d(p, query) < r

So, here is the broad setting. That suppose you have a set of data points and a query q, and this data points will be given to you beforehand. And, you have the answer the question of what is the nearest data point to the query. The query is not known to you right you can pre-process the data points before hand.

And you have to answer the question of what is the nearest data point to the query. While this question is itself is not very well post at because, what are we asking for are we? Are we asking for the K-nearest neighbors or are we given K or are we given some radius r and we are asking for the set of the all the set of points that are within distance r to the query q. And, what is the distance function right all of these need to be defined, but before we go head and defined these, let us see what are some examples of this what are some use cases of this particular problem right.

(Refer Slide Time: 01:41)



Applications

- Numerous
  - Finding near duplicate webpages / articles
  - Finding similar images in search
  - Clustering
  - Nearest neighbour classifier
- Variants
  - all pairs near neighbors

IIT Gandhinagar Indian Institute of Technology Gandhinagar

NPTEL ONLINE CERTIFICATION COURSES

3

It turns out that there are numerous applications right as there is; obviously, whole lot of applications in such, right. Suppose you are you are sort of designing a such engine in which a in which you given image and you are asking for similar images ok. There is also are large application in d duplicating web search right. And one of the major applications that I mean in this such engines is that in the web what you would see is that there are multiple copies of a webpage.

Right because, may be it is a news article right, may be it is a news article from associated press and the same news article at basically very, very similar news article have appeared in the I mean slide modified versions of at, or even the same article with some with some ads on top of it has appeared in multiple pages right.

And as sort of nice search engine, you would not want your front page you would not want your search page to be loaded with multiple copies of the same article. Because these it does not get much I have seen multiple copies of same article. So, what all this search engines to and it also takes tuple of modes space, it also takes have a lot more sort of resource in terms of crawling banned which resource names of crawling storage basic terms of storing multiple copies of the same article and how will lot of confusion right.

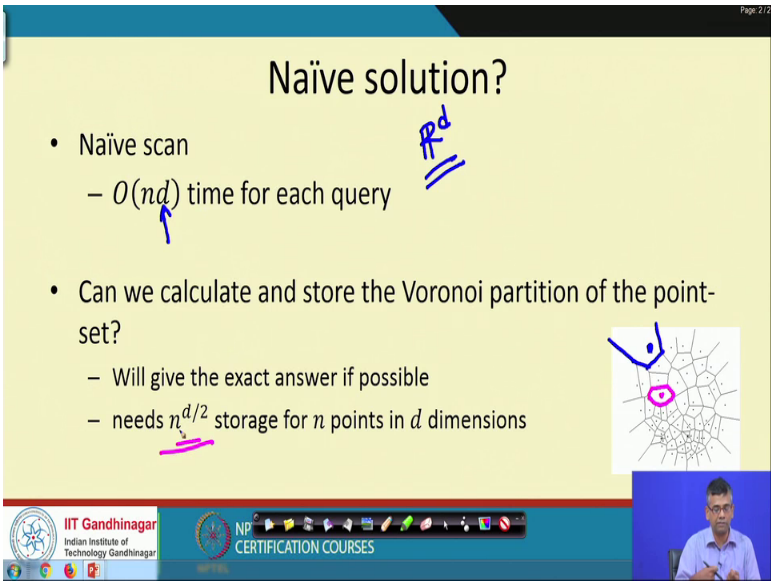
Plus, it might also confused you the ranking because now I mean all these articles all the different copies if you regard them different articles they are pointing to the all there out links are appointing to the same sort of webpages. So, the page rank with modified and so on right. So, you want to do d duplication right. And for various reasons this is not exact d duplication. Because, two webpages the might have just straightly different html they might have straightly different ads or css or and so, it is not exact duplicates in terms of the in terms of the entire webpage content right.

So, this near duplication for web pages is a very big application of the of locality sensitive hashing. And we will go over it will see something related to that. Is also very natural question to asking clustering right. Because, what is clustering I mean after all rather than just clubbing the same items together right. And then and then we could kind of ask that given a particular, given a particular no, given a particular data point, what are the items are the close to it right, that is what we are asking there is an obvious application in designing at nearest neighbor classified.

So, nearest neighbor classified is basically one in which given a test point right, you would look at the neighbor surrounded. This neighbors you would look at the labels of the neighbor surrounded and you would assign a label to the test point based on the labels of the neighbors. So, before you assign at a label you need to answer the question of how do I find the nearest neighbors efficiently right.

A slight variant of this problem which requires slightly different set of algorithms is the question of all pair near neighbors, in which rather than being given a particular query you are given a data set and you have to find out all pairs of all I mean all pairs of data points that are considered near neighbors.

(Refer Slide Time: 05:06)



Naïve solution?

- Naïve scan
  - $O(nd)$ time for each query
- Can we calculate and store the Voronoi partition of the point-set?
  - Will give the exact answer if possible
  - needs $n^{d/2}$ storage for n points in d dimensions

So, again what is the naive solution to this algorithm, right? Before we start looking at smarter solutions; so, the naive solution the most naive solution is of course, a naive scan right. That given the, that given a query at query time you do you sort of go for all the data points right. And if you assume that the data points are in are in Rd right, we have in explicitly mention that, but this is the, but this is the dimen this is the size of if you assume them to be vectors in Rd, then you need time order n which is the number of points times order d to compare each point with the query right.

So, that needs total time order n d. So, answer the nearest neighbor question. This is exact of course, this find it too much right, I mean n could potentially the, if you order sort of self-respecting data scientist you are you are dealing with n to be millions right. And d this also potentially large and that at test time you certainly do not have enough time to sort of do this exhaustive search; so, what can we do? The other extreme alternate I mean alternative is to compute the voronoi partition of the of the point set. So, what is the voronoi partition? So, the given a particular set of points voronoi partition is the following is the following is a following partitioning of the space.

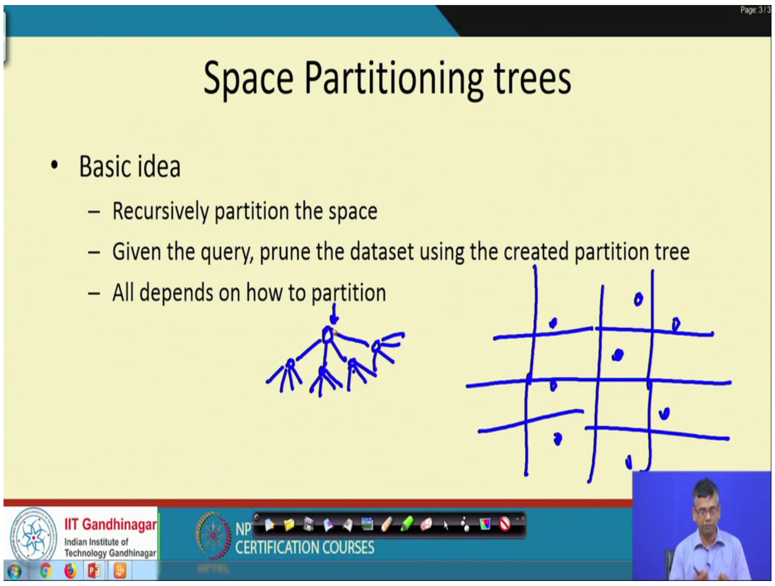
So, imagine that we are in r 2 right. And now we have marked of the region that of all of all points, that a closest to this particular blue point right. That is the voronoi cell for the blue point right. And similarly we do this for every for every other point. I mean I can do this for this for this let us call this a pink point right.

Then I mark of this cell right and so on right. So, of course, the voronoi partition depends on the on the other points that are there in the data right, but once if you variable to do this, once a query point comes you just need to check which voronoi partition at falls n and that would give you the exact answer; however, life is not as smooth.

Right so, the voronoi partition into in 2 dimensions is fine, but the voronoi partition in d dimension needs a lot of storage. In fact, it can need storage of the order of n to the power d by 2 right; which is not nice right I mean because the I mean while we do not want to spend time order n d, we can no also not spend time and time or space that is n to the power d that is even was.

And because once you store the voronoi partition you I mean given a query point you also have to calculate it I mean yeah I mean that is really the number of phases of the voronoi partition. So, given a query point you have to compare it with each of these phases and so on that is that is right too much. So, I do not want to do this ok.

(Refer Slide Time: 08:09)



Space Partitioning trees

- Basic idea
  - Recursively partition the space
  - Given the query, prune the dataset using the created partition tree
  - All depends on how to partition

IIT Gandhinagar
Indian Institute of Technology Gandhinagar
NPTEL CERTIFICATION COURSES

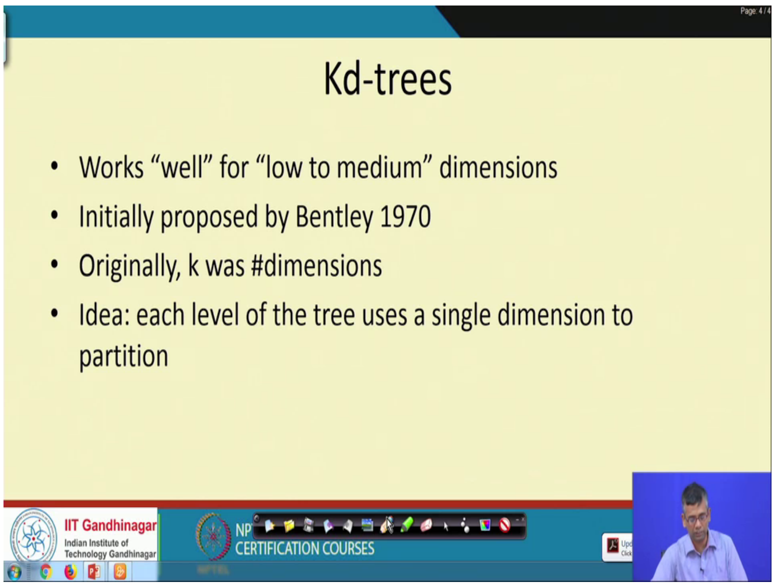
So, one set of techniques that will look at it this lecture are calls space partitioning trees. So, what is a space partitioning tree? The intuitive idea is very simple actually, is says that let us recursively partition given the points right suppose given the points let us recursively partition the space right.

So, let us say that we first partition it according to I mean according to the x and the y axis right. And then we and then we and then we and then we again partition it according to here, then we again partitioning according to here then we again partition it here then we again partitioning it here right. So, I am a calling with a tree, I am calling into tree because you can imagine that the top level partition had 4 children right corresponding to the 4 axis. Then each of these partitions again has 4 children corresponding to the 4 axis that it creates and so on right; so, this right.

And then and then given a query given a query point, you just for every note you just ask which of these children do I need to go down into right. So, this is a broad framework right. The partitioning method that I showed you is really the most naive one right. This broad framework can be made concrete in a bunch of different ways and we will see a couple of different ways right.

And this is also bunch of ways to search for a given query right. The most critical thing the most critical step in this in this frame work is to decide how to do the partition. And we will see a few ways of how to do this partition.

(Refer Slide Time: 09:58)



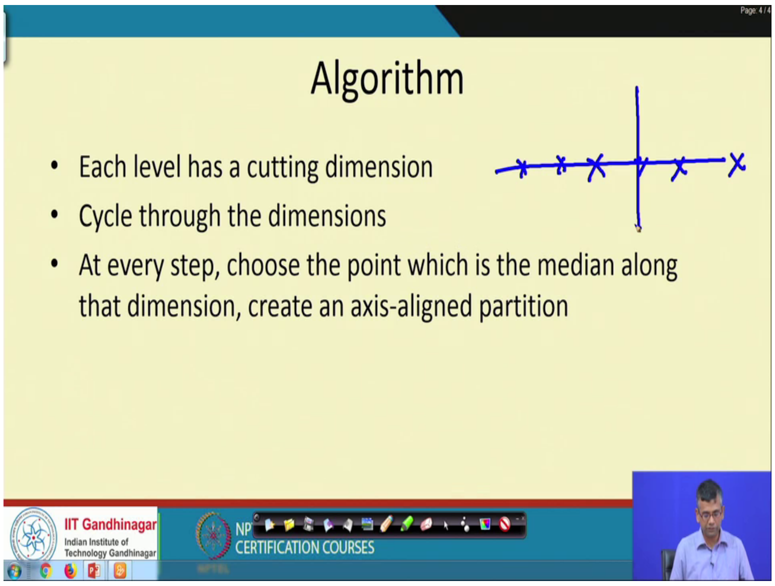
Kd-trees

- Works "well" for "low to medium" dimensions
- Initially proposed by Bentley 1970
- Originally, k was #dimensions
- Idea: each level of the tree uses a single dimension to partition

So, one of the very, very common ways that is that is very sort of much used even today is known as k d trees right. So, this was initially proposed by Bentley in around 1970. And when it was initially developed I mean the number k is denote the number dimensions. So, it used to be called 2D trees, 3D trees and so on.

But now it has just coming to the name. So, what is the intuition here? The intuition here is at each level of the tree uses a single dimension to partition ok. How does it choose the dimension what is the value in the dimension all that will come right? So, so let us see an. So, let us see the algorithm right.

(Refer Slide Time: 10:41)



Algorithm

- Each level has a cutting dimension
- Cycle through the dimensions
- At every step, choose the point which is the median along that dimension, create an axis-aligned partition

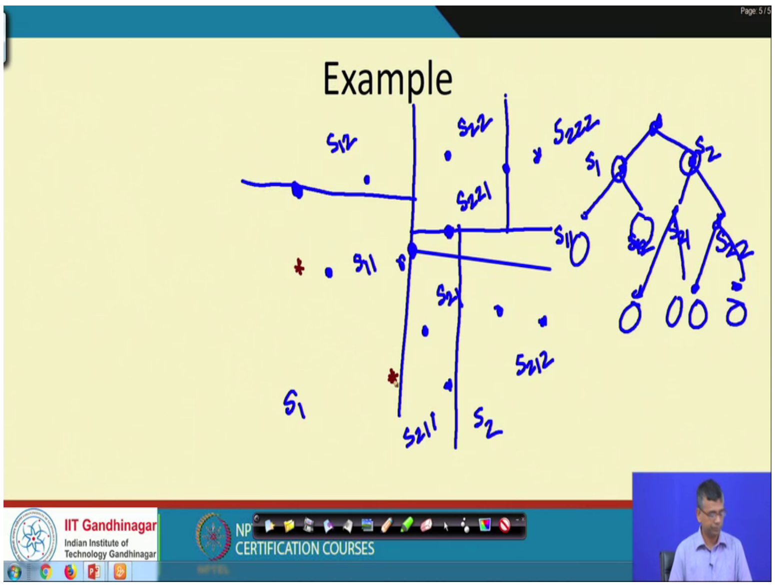
So, we start with the entire set of points with each level you associate a cutting dimension. Let us say you have like 3 dimensions. So, what we will try to do is cycle through this dimensions. Essentially let us say we start of the first dimension and say we from the first cut according to some threshold in the first dimension. Then the second level at the second level we use the second dimension right. And then we and then we choose some threshold in the some hyper plane was some plane in the second dimension to cut the second to make the second level children and so on right.

Now, how do I choose it threshold? What we would do is as follows at every step right; we would try to balance of the tree right. We would try to keep the number of left children equal to the more or less equal to the number of right children. And the way to do that would be to say that you have considered and dimension right, you have a set of points that you want to partition along this dimension. Now, look at the value the coordinates values of the points along this dimension.

So, we have this dimension that you have chosen. You have the points that you want to partition right. And look at the coordinate values of these points along this dimension take the median coordinate value, and use that as the threshold in this dimension. So, basically then you put points to the left all the points to the left of the median in one of a in one child all the points to the right are median in one child. So, this would be k d tree with 2 children per note.

And this would and we would create an axis aligned partition that uses the median value in every in each of the dimensions ok. So, let us do a small example maybe um. That is your small example. That suppose we have let us. So, small example in r 2 itself, right.

(Refer Slide Time: 12:44)


Example

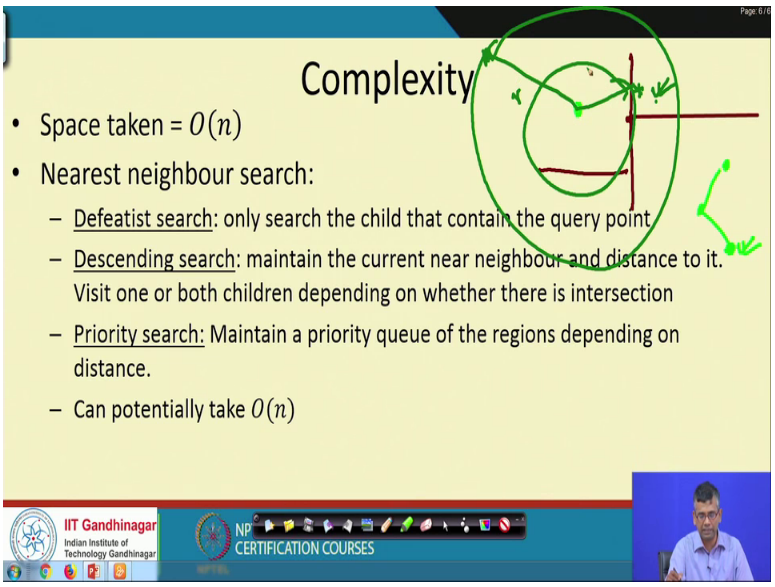
So, maybe first we choose the dimension we choose the partition along this dimension. So, this is my first cut next maybe. So, my first tree is this and then we go in left and right. S1 S 2 then, we partition we have this set of points S1 right now you need to partition S1 right. Then we choose the median here, let us say that the median here is this right. So, this would be the cell S1 1 this would be the cell S1 2 right

Now, here the median partition might be this. So, right may be S1 1 and S1 2 we do not want to partition anymore. Because they have only one point, but may be here you want a partition. Suppose we had some few more points here. Maybe here you want a partition according to this here may be you want a partition according to this. So, this is S 2 2 1 S 2 2 2 S 2 1 1 S 2 1 2 some just numbering is arbitrarily according to some notation that I have the numbers do not really mean much in the labels of the partitions.

So, at these points; so now, we have the partitions and in the leaves you store the data points right. And you could there is also data point associated with every node right because, you are using the medians. So, you could say that the inner node S1 stores the data point stores the data point this one. This, right the inner node S 2 stores this data point right. And so on right the first root note stores the data point this stores this root data point.

So now given a particular query, what would be what would we do? That is the question right. So, suppose a given a query here what would we do right. Or maybe if you or may be the query lies here what should we do right. And there are different things that you can do here right and let us go over this some of this.

(Refer Slide Time: 15:45)



Complexity

- Space taken = $O(n)$
- Nearest neighbour search:
  - Defeatist search: only search the child that contain the query point
  - Descending search: maintain the current near neighbour and distance to it. Visit one or both children depending on whether there is intersection
  - Priority search: Maintain a priority queue of the regions depending on distance.
  - Can potentially take $O(n)$

Page: 6 / 6
IIT Gandhinagar
Indian Institute of Technology Gandhinagar
NPTEL ONLINE CERTIFICATION COURSES

So, the space taken as clearly order n, for the data structure because you sort of you do not replicate anything you just divide the entire set of points if you can dividing and that is it. So, there are typically 3 strategies for nearest neighbours search in k d trees. So, the first strategy is known as a defeatist search. What you do there is that you only search the child that contains a query point right. What does it mean?

What that means is that, suppose you have divided the, suppose you have divided according to here right and then suppose your query point falls your query point is here falls here right. So, then from the route you would only go to the left, and from this child you would again go the right. So, route you go to the left and then you go to the right.

You search you search only in this, but as you can imagine that this might not be entirely correct. Because it is possible for instance that the nearest neighbour to this to this particular point lies somewhere here right. Maybe there is nothing I mean maybe there is one point in the cell, but maybe that point is very far of here right.

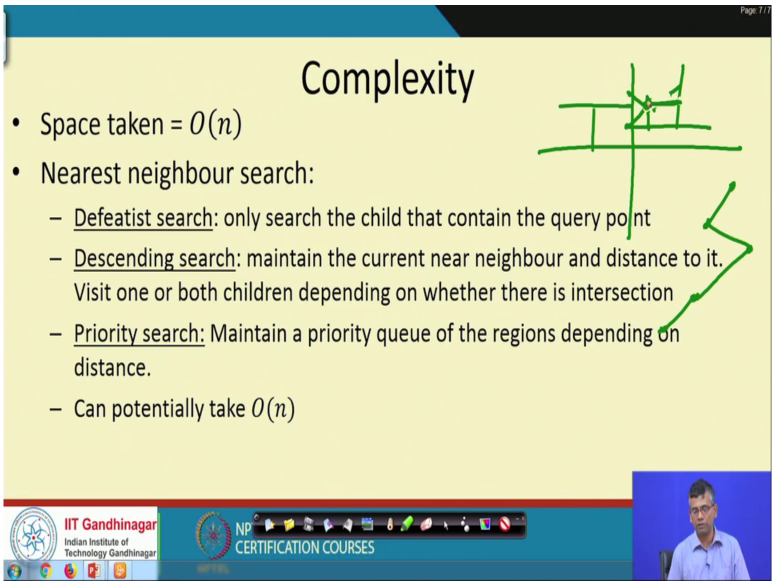
So, just because a query point falls in this cell, does not mean that the nearest neighbour will be in this cell right. Because the because the nearest because a cells because a query point might be close to some boundary of the cell, governing nearest neighbour I mean ideal place to that nearest neighbour would be somewhere else, but the, but what the defeatist search does is that is just is just I mean is does not considered that. Is just keeps on going and then and then if it does not find anybody you would finds anybody in the cell it returns that which might be wrong or it just gives up if you does not find anybody this cell.

So, that is it; so, then the descending search right. What it for do is that it would say that let me see, let me go down this cell right and let me maintain the current nearest neighbour and the distance to it. So, for instance it would say that I have gonna down this cell and then my candidate nearest neighbour is this. So, it maintains the distance to it. Then it looks at the path that is has followed right. Functions it when it when left here it when right here and then it says that if this is the radius, if I consider a ball of this radius right does it intersect which are the cells does it intersect right. Then it sees that intersect this other cell and therefore, I must go up my recursion tree and explore this other cell also ok. So, it does that.

So, basically and then it is gets this potential nearest neighbour as a candidate. And then it is says that let me consider now this new ball. So, which are the cells does it intersect this is a bad drawing, but you get the idea right. So, it keeps on exploring the it sort of sees it has a candidate nearest neighbour near neighbour. It and the candidate distance to it sees how many it construct the ball around the query point and sees which of the cells does it intersect. And then it goes and explores those selves also by winding up the recursion. And therefore, once it explores it the candidate near neighbour gets a little closer because a my change the candidate. And so, the candidate distance also that it is closer.

So, this is this is definitely going to give you the nearest neighbour right, but it might end of searching entire tree. A slightly better slightly better combination of these 2 is what is known as a priority search. In which case it does what it does is that as it is going down the tree.

(Refer Slide Time: 19:45)



Complexity

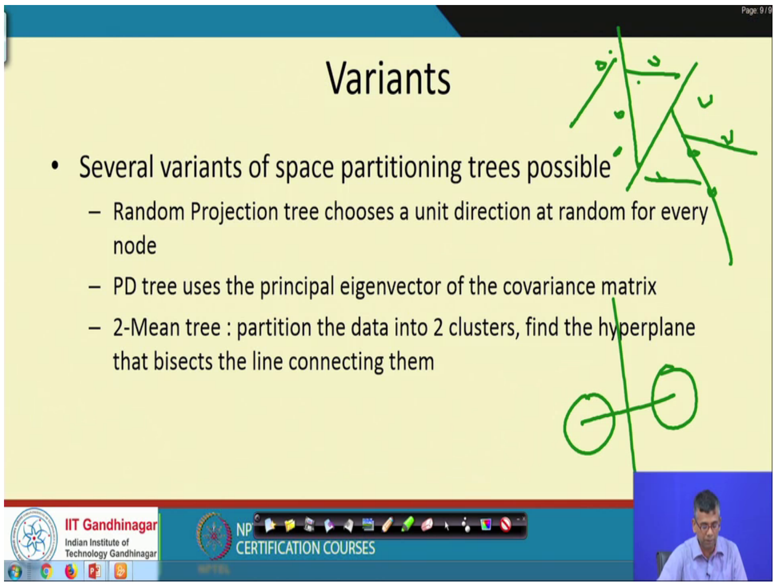
- Space taken = $O(n)$
- Nearest neighbour search:
  - Defeatist search: only search the child that contain the query point
  - Descending search: maintain the current near neighbour and distance to it. Visit one or both children depending on whether there is intersection
  - Priority search: Maintain a priority queue of the regions depending on distance.
  - Can potentially take $O(n)$

May be as it is going down the tree, the query point maintains a priority over the over the over the regions depending on the distance. For instance is says that this I have I have gone down this tree to here, here, here, here, here, but now if I look at all the particular partitions that all the cells that have seen, my distance to the cell is this much my distance to this particular cell is this much and distance to the cell is this much my distance to this cell is this much right. On this it is this it is this right.

And so, it gives the priority queue over all this over all this cells right. And then it decides to visit the cells based on the priority queue right. So, depending on how you implement the priority queue and where you terminate the search this can also return you I mean guarantee you that the that the nearest neighbour will be returned.

However, again in the worst case you can potentially end up searching all the nodes. And that is the basic problem with the k d trees right. And it does not do a very good job of answering the nearest neighbour queries in the in the case of in the in the worst case right.

(Refer Slide Time: 21:13)



Variants

- Several variants of space partitioning trees possible
  - Random Projection tree chooses a unit direction at random for every node
  - PD tree uses the principal eigenvector of the covariance matrix
  - 2-Mean tree : partition the data into 2 clusters, find the hyperplane that bisects the line connecting them

So, there are several variants of space partitioning trees possible as I mentioned right. And here are a few of the more important once. There is something called a random projection tree right. What does it do? Remember that we are creating a partition at every step right what it does is that when you search of the data right, let us say let us say that this is the let us say that this is the data right this is the data it in sort of takes the random hyperplane and tries to partition the and partitions the data as like either on the left of it or on the right of it. And then, it taking once it does the partition and this will approximately divide the data into half and again once, it does it this I mean this takes another random hyperplane this takes another random hyperplane and tries to partition the data.

And then this take another random hyperplane they this takes another random hyperplane here another one here another one here and tries to partition the data right. And that is how it is keeps on growing the left on the right the right sub tree of any of any node. Ok this is known as the RP tree or a random projection tree. I mean again the variant of this is that you could use any other direction right, and you could use something like the principle eigenvector of the covariance matrix right. What is intuitions here? The intuition is the here is that the direction of the principle eigenvector of the covariant matrix is the one along which you can partition the data the most. It is the one with direction of the maximum variance.

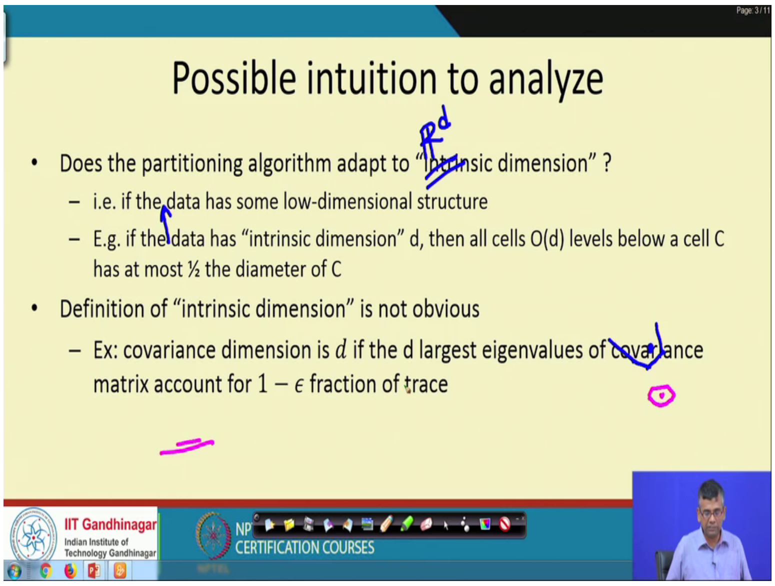
So, therefore, if you half a hyperplane that is orthogonal to the that direction that cuts the data, that is separates the data the most in some in some surface to volume kind of ratio right. That I mean we can make this a little more precise, but in a surface to volume sense you are you can if you take the direction that is orthogonal to the take the hyperplane that is orthogonal to the to the principle eigenvector, you get you get a good direction to separate the data right. And you can keep on doing that.

So, you at every node; so, first you take the entire data set calculate the principle eigenvector and partition according to that. Then you have the left sub tree have the right sub tree. Again you calculate the covariance matrix of these two, you again take the principle eigenvectors and then and then and then separate and keep on doing this. So, this is a little expensive right. It is it performs not I mean the performance is not bad.

It is a little it is not trivial to analyse and it is also little expensive because at every time point you calculating the principle eigenvector. Then there is 2 mean tree right which says that let us calculate the let us not exactly calculate the principle eigenvector. Let us try to do 2 clustering of the data right. Which is effectively other principal eigenvector trying to do. Let us try to put some to the left I mean let us try to run came in with k equal 2 for instance and we take the centres, we take the 2 centres and we take the line separating the 2 centres.

So, is the 2 centre is the clusters will look like this, we take the 2 centres we take the orthogonal direction of the orthogonal hyperplane to the line connecting at 2 centres and we use that the separate the data right. So, the data has nice cluster structure this will this will of course, what nicely. So, the problem is that it is very hard to quantify the worst case performance of any one of them of any of these.

(Refer Slide Time: 24:38)



Possible intuition to analyze

- Does the partitioning algorithm adapt to "intrinsic dimension"?
  - i.e. if the data has some low-dimensional structure
  - E.g. if the data has "intrinsic dimension" d, then all cells O(d) levels below a cell C has at most ½ the diameter of C
- Definition of "intrinsic dimension" is not obvious
  - Ex: covariance dimension is d if the d largest eigenvalues of covariance matrix account for $1-\epsilon$ fraction of trace

$\mathbb{R}^d$
IIT Gandhinagar
Indian Institute of Technology Gandhinagar
NPTEL CERTIFICATION COURSES

However, there are possible ways to try to analyse these right. And these and these and these particular techniques they all can be analysed in terms of this common thing right. You could try to ask there are if I use a space partitioning tree does the partitioning algorithm adapt to the intrinsic dimensionality of the data.

So, what do I mean by intrinsic dimensionality? For instance, so, imagine that you have you have you have piece of paper in that you are holding up in 3 dimensions. So, although and you take points on this piece of paper right. So, although the dimensionality this points have a 3 dimensional these are points in 3 dimensions. They really lie on this on this 2 dimensional plane right. Similarly, what might happened is that although the data is been represented in high dimension. The effective dimension somehow it more or less lies in a low dimensional plane sub space.

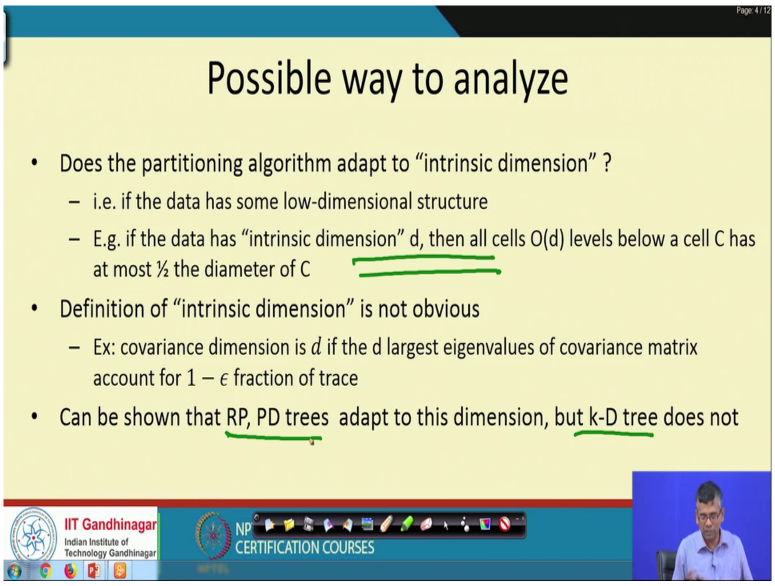
So, what we would really like is that if the data lies in some low dimension really, in that I mean the partitioning algorithm should really depend on this intrinsic dimensionality of the data right. It is not I mean an here is one way to 2 kind of formalize this right. What we might want to says is that if the intrinsic dimension is d. So, note that haven't really defined to you what intrinsic the dimensionality is, but if the intrinsic dimensionality is d then after ordered d levels, and why ordered d because your cycling through all the dimensions that after we are cycles.

So, all the dimensions we substantially shrink the size of each cell right. That the size of itself becomes something like half of what it was at this step. So, is that happens then I can says that the depth of the tree something like d log the maximum diameter of the of the data set right, d times the log of the diameter of the original data set right which means that the it is of it is re-establishing small.

So, in order to formalize this, you need to know what is the I mean how to sort of formalize intrinsic dimension. There are couple of phase to do this the something (Refer Time: 26:53) dimension. So, here is one sort of easy to say to way to do this right. We see that the intrinsic dimensionality sometimes we say that the intrinsic dimensionality of a data set is d right. If you look at the d largest eigenvalues of the covariance matrix and they account for 1 minus epsilon fraction of the trace.

So, remember that the trace is a some of the eigenvalues right. So, if the if taking if the original dimension is capital D, but only small d of the; so, think of small d as let us say let us say 10 percent of capital D. Evenly small d of the largest eigenvalues account for more or less the entire trace let us say 99 percent of the trace then will say that intrinsic dimensionality is small d. And it is known. So, why is this definition important? Because, it is known that at least for some of the at, least for some of the algorithms right.

(Refer Slide Time: 27:46)



Possible way to analyze

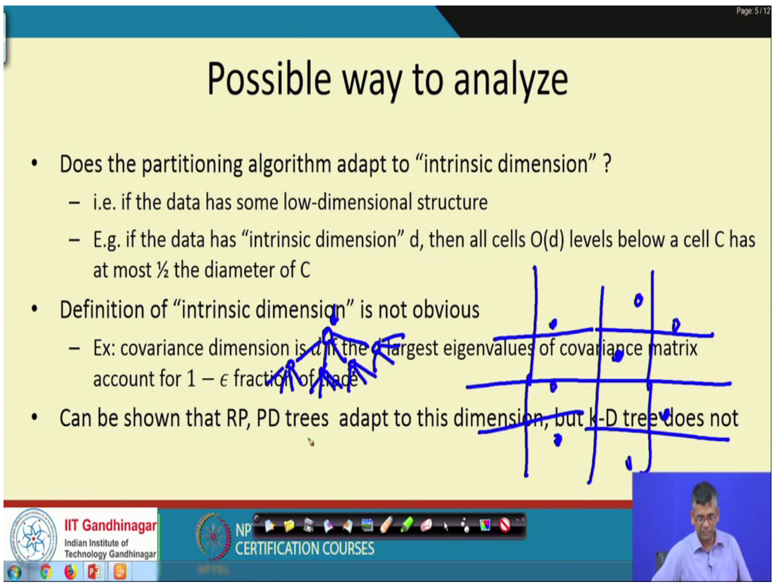
- Does the partitioning algorithm adapt to "intrinsic dimension" ?
  - i.e. if the data has some low-dimensional structure
  - E.g. if the data has "intrinsic dimension" d, then all cells O(d) levels below a cell C has at most ½ the diameter of C
- Definition of "intrinsic dimension" is not obvious
  - Ex: covariance dimension is d if the d largest eigenvalues of covariance matrix account for $1 - \epsilon$ fraction of trace
- Can be shown that RP, PD trees adapt to this dimension, but k-D tree does not

Page: 4 / 12
IIT Gandhinagar
Indian Institute of Technology Gandhinagar
NPTEL CERTIFICATION COURSES

for instance the RP the random projection and the principle direction trees adapt to this definition of the dimension. That is, they have this particular property. They have they have this particular property. When you define the intrinsic dimension in terms of the covariance matrix, but something like the k d trees does not adapt to the intrinsic dimension because it is always using the median.

So, it is not really using the structure any deeper structure of the of the data right. So, in general the rule of thumb is that if you are really interested in working with space partitioning trees it might be useful to look at RP trees and principle direction trees. Unless of course, you have you have library implementation k d trees and you do not want implement anything on your own ok.

(Refer Slide Time: 28:33)



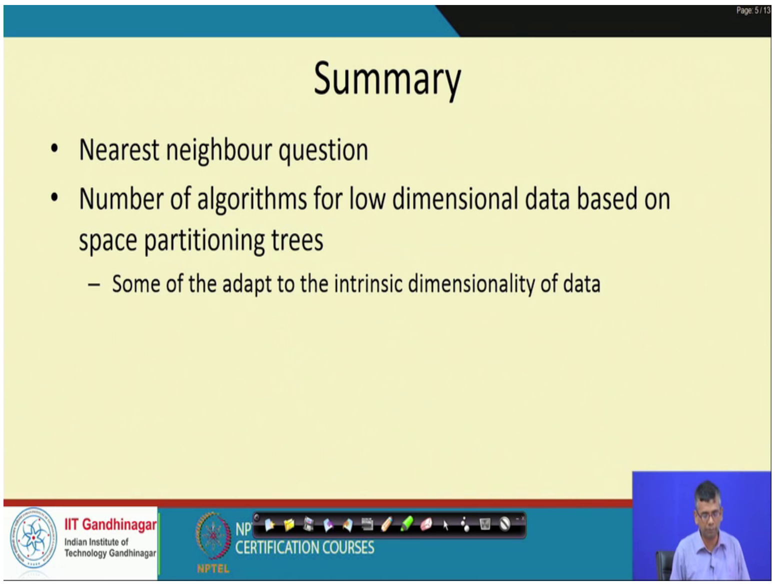
Possible way to analyze

- Does the partitioning algorithm adapt to "intrinsic dimension" ?
  - i.e. if the data has some low-dimensional structure
  - E.g. if the data has "intrinsic dimension" d, then all cells O(d) levels below a cell C has at most ½ the diameter of C
- Definition of "intrinsic dimension" is not obvious
  - Ex: covariance dimension is d if the d largest eigenvalues of covariance matrix account for $1-\epsilon$ fraction of space
- Can be shown that RP, PD trees adapt to this dimension, but k-D tree does not

IIT Gandhinagar
Indian Institute of Technology Gandhinagar
NPTEL CERTIFICATION COURSES

But, but these are, but the, but the RP trees and the RP trees and the PD trees are almost always better performing than the ordinary k-D tree.

(Refer Slide Time: 28:43)

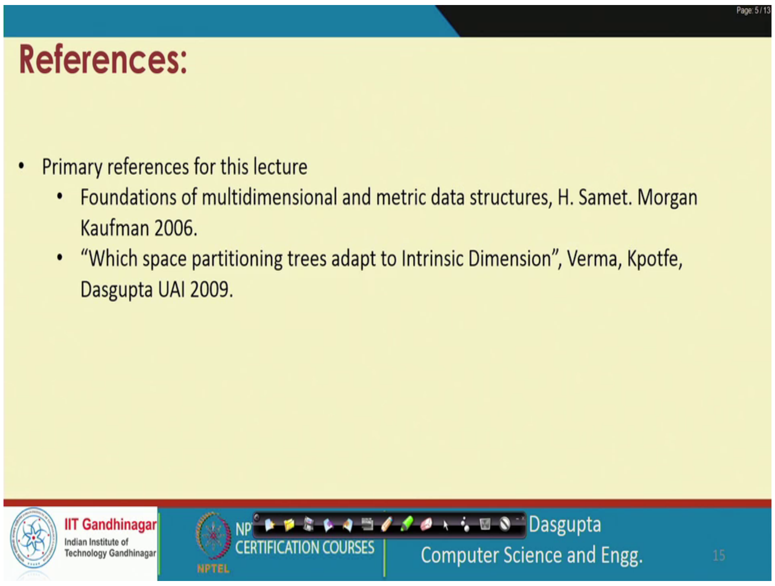


Summary

- Nearest neighbour question
- Number of algorithms for low dimensional data based on space partitioning trees
  - Some of the adapt to the intrinsic dimensionality of data

So, just to summary just to summarize who introduce a nearest neighbour question in this lecture. We also looked at family of algorithm space on space partitioning trees, and the and the rule of thumb is that if you are trying to choose one out of this family, if you try to choose one that adapts to the intrinsic dimensionality of the data right. And 2 of the ones that we saw and 2 of the ones that I mentioned are random projection trees and principal direction and principle direction trees.

(Refer Slide Time: 29:11)



References:

- Primary references for this lecture
  - Foundations of multidimensional and metric data structures, H. Samet. Morgan Kaufman 2006.
  - "Which space partitioning trees adapt to Intrinsic Dimension", Verma, Kpotfe, Dasgupta UAI 2009.

So, if you want to look into more into some of these, there is this a book by Professor Samet on foundations of multidimensional and metric data structures. There is also bunch of very nice papers by Professor Sanjay Dasgupta. And in particular this particular paper talks about the notion of intrinsic dimension and how it applies to the different phase partitioning trees. So, feel free to look at that, and that is it for this lecture.

Thank you.