

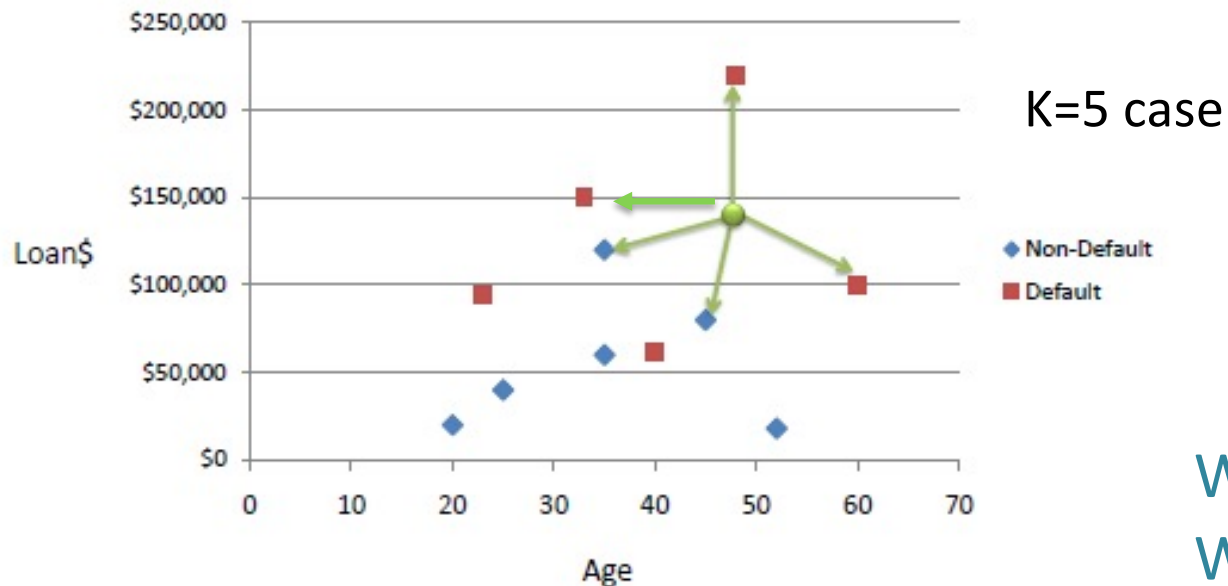


K Nearest Neighbor Classification

plus similarity and distance calculations

K Nearest Neighbor Classification

- K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions).
- A case is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its **K** nearest neighbors measured by a distance function.



What if $k=1$?
What if $k=4$?

K Nearest Neighbor Classification

- Two concerns:
 - How to compute the distance between two data objects?
 - What is an efficient way to find the K closest data objects (i.e., neighbors)?

Compute Distance between Data

Data Types:

- Interval-scaled variables
- Binary variables
- Nominal, and ordinal variables
- Variables of mixed types
- Text
- Temporal

Standardize Numeric Data

- Why do we need to standardize/normalize numeric data?
- Standardize data

- Calculate the mean absolute deviation:

$$s_f = \frac{1}{n} (|x_{1f} - m_f| + |x_{2f} - m_f| + \dots + |x_{nf} - m_f|)$$

Where

$$m_f = \frac{1}{n} (x_{1f} + x_{2f} + \dots + x_{nf}).$$

- Calculate the standardized measurement (**z-score**) $z_{if} = \frac{x_{if} - m_f}{s_f}$

- Normalizing data $z_{if} = \frac{x_{if} - m_f}{\sigma_f}$

Similarity/Dissimilarity Between Objects

- Distances are normally used to measure the similarity or dissimilarity between two data objects
- Some popular ones include: *Minkowski distance*:

$$d(i, j) = \sqrt[q]{(|x_{i1} - x_{j1}|^q + |x_{i2} - x_{j2}|^q + \dots + |x_{ip} - x_{jp}|^q)}$$

Where $i = (x_{i1}, x_{i2}, \dots, x_{ip})$ and $j = (x_{j1}, x_{j2}, \dots, x_{jp})$ are two p -dimensional data objects, and q is a positive integer

- If $q = 1$, d is *Manhattan distance*

$$d(i, j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \dots + |x_{ip} - x_{jp}|$$

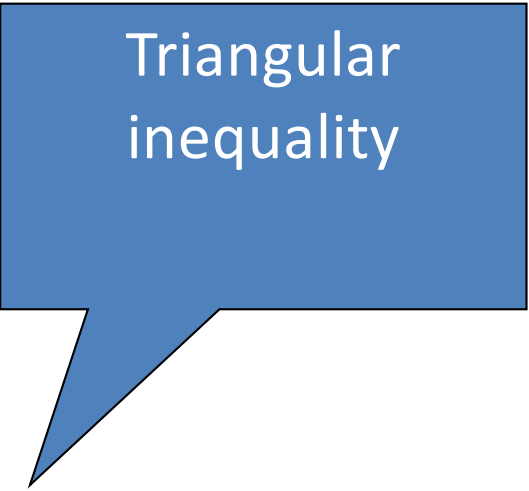
Similarity/Dissimilarity Between Objects

If $q = 2$, d is Euclidean distance:

$$d(i, j) = \sqrt{(|x_{i_1} - x_{j_1}|^2 + |x_{i_2} - x_{j_2}|^2 + \dots + |x_{i_p} - x_{j_p}|^2)}$$

- Properties

- $d(i, j) \geq 0$
- $d(i, i) = 0$
- $d(i, j) = d(j, i)$
- $d(i, j) \leq d(i, k) + d(k, j)$



Triangular
inequality

Other Similarity/Distance measures

- **Sets as vectors:** measure similarity by the cosine distance.

$$x_i = [x_{i1}, x_{i2}, \dots, x_{ip}]$$

$$x_j = [x_{j1}, x_{j2}, \dots, x_{jp}]$$

$$\cos(x_i, x_j) = \frac{x_i \bullet x_j}{|x_i| * |x_j|} = \hat{x}_i \bullet \hat{x}_j$$

Similarity/Dissimilarity for Binary Data

- **Symmetric attribute:** both states are equally valuable, carrying the same weight, e.g., gender 0, 1 \rightarrow coding each state to be 0 or 1 arbitrarily
- **Asymmetric attribute:** outcomes of the states are not equally important, e.g., outcome of a medical test \rightarrow positive or negative
 - assign the more important outcome to value 1
 - assign the less important outcome to value 0
 - e.g., 1 : HIV positive, 0: HIV negative. The agreement of two patients having “1”s for this attribute is more significant than an agreement of “0”s.

Similarity/Dissimilarity for Binary Data

- A contingency table for binary data

		Object j		
		1	0	sum
Object i	1	a	b	$a+b$
	0	c	d	$c+d$
	sum	$a+c$	$b+d$	p

- Simple matching coefficient (if the binary variable is symmetric):

$$d(i, j) = \frac{b+c}{a+b+c+d}$$

- Jaccard coefficient (if the binary variable is asymmetric):

$$d(i, j) = \frac{b+c}{a+b+c}$$

Dissimilarity between Binary Variables

- Example

Name	Gender	Fever	Cough	Test-1	Test-2	Test-3	Test-4
Jack	M	Y	N	P	N	N	N
Mary	F	Y	N	P	N	P	N
Jim	M	Y	P	N	N	N	N

- gender is a symmetric attribute
- the remaining attributes are asymmetric binary
- let the values Y and P be set to 1, and the value N be set to 0

		Jack	
Mary		1	0
	1		
	0		

		Jack	
Jim		1	0
	1		
	0		

		Jim	
Mary		1	0
	1		
	0		

Dissimilarity between Binary Variables

Jaccard
coefficient

$$d(\text{jack}, \text{mary}) = \frac{0 + 1}{2 + 0 + 1} = 0.33$$

$$d(\text{jack}, \text{jim}) = \frac{1 + 1}{1 + 1 + 1} = 0.67$$

$$d(\text{jim}, \text{mary}) = \frac{1 + 2}{1 + 1 + 2} = 0.75$$

Gender is not yet included in the computation

		Jack	
Mary		1	0
	1		
	0		

		Jack	
Jim		1	0
	1		
	0		

		Jim	
Mary		1	0
	1		
	0		

Nominal Attributes

- A generalization of the binary attribute in that it can take more than 2 states, e.g., red, yellow, blue, green
- Method 1: Simple matching
 - m : # of matches, p : total # of variables

$$d(i, j) = \frac{p - m}{p}$$

- Method 2: use a large number of binary attributes **Hot code encoding**
 - creating a new binary variable for each of the M nominal states

Ordinal Attributes

- An ordinal attribute can be discrete or continuous
- order is important, e.g., rank
- Can be treated like interval-scaled
 - replacing x_{if} by their rank $r_{if} \in \{1, \dots, M_f\}$
 - map the range of each attribute onto $[0, 1]$ by replacing i -th object in the f -th attribute by

$$z_{if} = \frac{r_{if} - 1}{M_f - 1}$$

- compute the dissimilarity using methods for interval-scaled attributes

Attributes of Mixed Types

- A database may contain different types of attributes
 - symmetric binary, asymmetric binary, nominal, ordinal, and interval.
- How to combine the dissimilarity/distance from data of a mixture of types?

Attributes of Mixed Types

- Use weighted formula to combine their effects.
 - Feature value missing, or asymmetric binary with
$$x_{if} = x_{jf} = 0 \quad \rightarrow \quad \delta_{ij}(f) = 0$$
 - Otherwise $\rightarrow \delta_{ij}(f) = 1$
 - feature is interval-based: use the normalized distance ($\delta_{ij}^{(f)}$: weight on feature f)

$$d(i, j) = \frac{\sum_{f=1}^p \delta_{ij}^{(f)} d_{ij}^{(f)}}{\sum_{f=1}^p \delta_{ij}^{(f)}}$$

- feature is ordinal
 - compute ranks r_{if} and
 - and treat z_{if} as interval-scaled

$$z_{if} = \frac{r_{if} - 1}{M_f - 1}$$

Practice Question

- Compute the distance between (obj1, obj2),

	Gender	Age	Heart Rate	Fever	Cough	Category
Obj1	F	18	120	N	N	Severe-1
Obj2	M	36	89	N	N	Normal

For Age: $m=42$, $s=3.5$,

For heart rate: $m=95$, $s=10$

Possible values for Category include : Normal, Severe-1, Severe-2, Dying

For simplicity in demonstration, use Manhattan distance for interval data.

Other Similarity/Distance measures

- Measure distance between words/address/query, or between DNA sequences by edit distance
 - Given two strings S_1 and S_2 , the minimum number of operations to convert one to the other
 - Operations are typically character-level
 - Insert, Delete, Replace, (Transposition)
 - E.g., the edit distance from **dof** to **dog** is 1
 - From **cat** to **act** is 2 (Just 1 with transpose.)
 - from **cat** to **dog** is 3.
- Generally computed by dynamic programming.

Edit distance

- Given two strings S_1 and S_2 , the **minimum** number of edit operations to convert one to the other
- Operations are typically character-level
 - Insert, Delete, Replace, (Transposition)
- E.g., the edit distance from *dof* to *dog* is 1
 - From *cat* to *act* is 2 (Just 1 with transpose.)
 - from *cat* to *dog* is 3.
- dynamic programming

Edit Distance

EDITDISTANCE(s_1, s_2)

```
1  int  $m[|s_1|, |s_2|] = 0$ 
2  for  $i \leftarrow 1$  to  $|s_1|$ 
3  do  $m[i, 0] = i$ 
4  for  $j \leftarrow 1$  to  $|s_2|$ 
5  do  $m[0, j] = j$ 
6  for  $i \leftarrow 1$  to  $|s_1|$ 
7  do for  $j \leftarrow 1$  to  $|s_2|$ 
8      do  $m[i, j] = \min\{m[i - 1, j - 1] + \text{if } (s_1[i] = s_2[j]) \text{ then } 0 \text{ else } 1, \text{fi},$ 
9           $m[i - 1, j] + 1,$ 
10          $m[i, j - 1] + 1\}$ 
11 return  $m[|s_1|, |s_2|]$ 
```

Figure 3.5 Dynamic programming algorithm for computing the edit distance between strings s_1 and s_2 .

An Example

What's the edit distance between two strings : **fast** and **cats**?

		f	a	s	t
	0	1	2	3	4
c	1				
a	2				
t	3				
s	4				

Practice Question

What's the edit distance between two strings : **Broco** and **Brunch**?

		b	r	o	c	o
	0	1	2	3	4	5
b	1					
r	2					
u	3					
n	4					
c	5					
h	6					

Weighted edit distance

- As above, but the weight of an operation depends on the character(s) involved
 - Meant to capture OCR or keyboard errors, e.g. *m* more likely to be mis-typed as *n* than as *q*
 - Therefore, replacing *m* by *n* is a smaller edit distance than by *q*
 - This may be formulated as a probability model
- Requires weight matrix as input
- Modify dynamic programming to handle weights

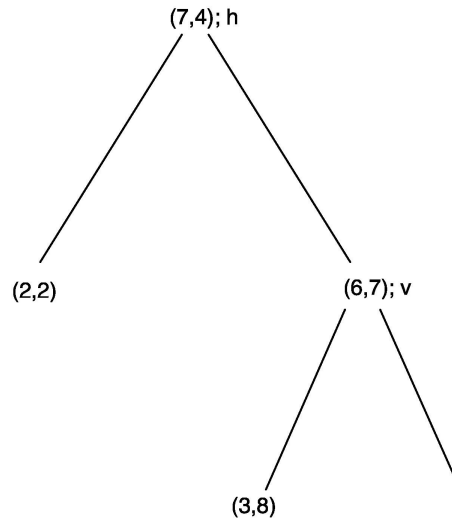
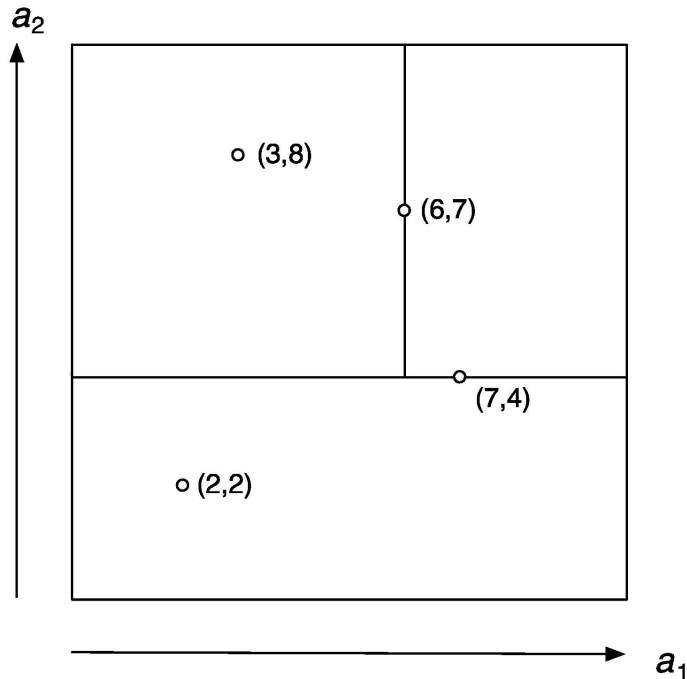
Finding nearest neighbors efficiently

- Simplest way of finding nearest neighbor: linear scan of the data
 - ♦ Classification takes time proportional to the product of the number of instances in training and test sets
- Nearest-neighbor search can be done more efficiently using appropriate data structures
- Two methods that represent training data in a tree structure:

kD-trees and *ball trees*

(k-dimensional tree)

k D-tree example (2D case)

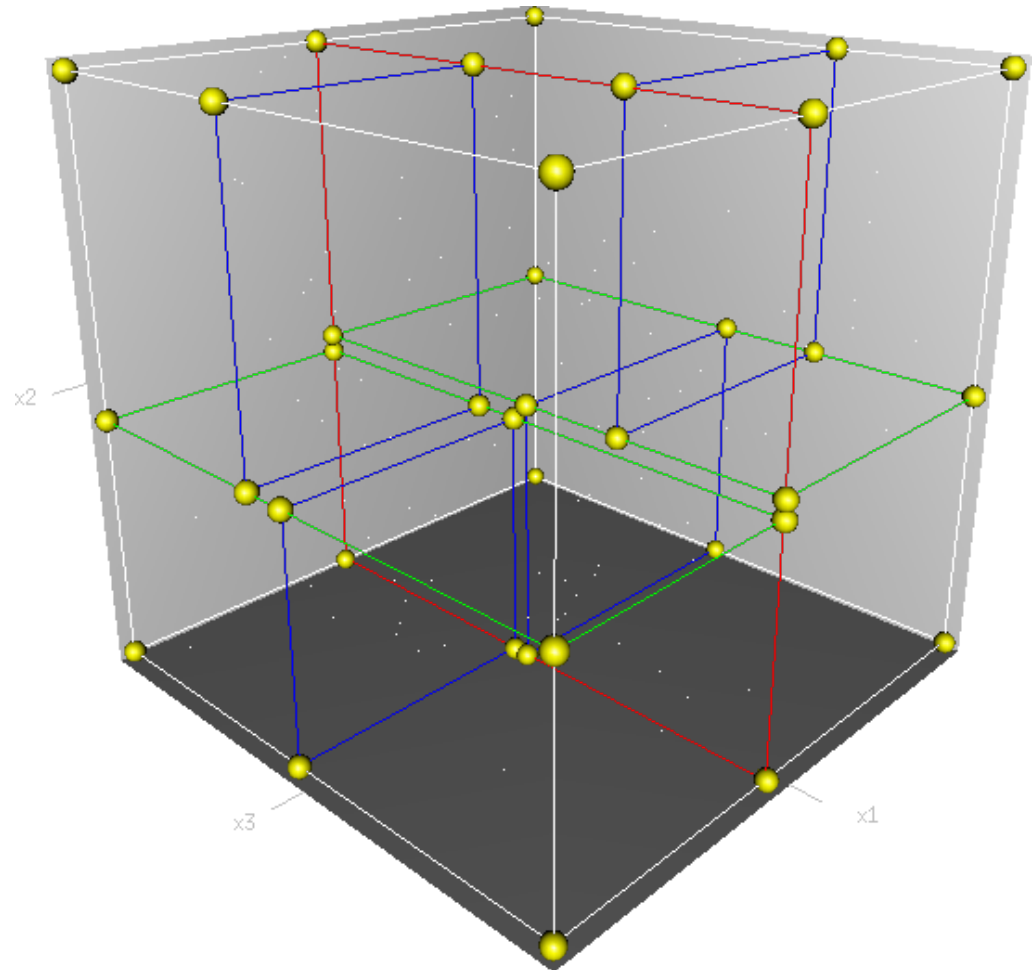


- Binary tree in which every node is a k -dimensional point.
- Every non-leaf node can be thought of as implicitly generating a splitting hyperplane that divides the space into two parts, known as half-spaces.

The hyperplane direction is chosen in the following way: every node in the tree is associated with one of the k -dimensions, with the hyperplane perpendicular to that dimension's axis.

k D-tree example (3D case)

- Binary tree in which every node is a k -dimensional point.
- Every non-leaf node can be thought of as implicitly generating a splitting hyperplane that divides the space into two parts, known as half-spaces.

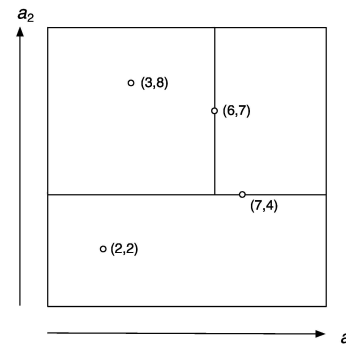
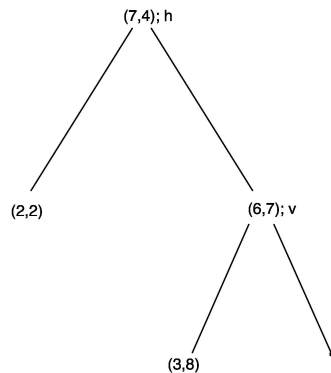


kD-tree example

```
function kdtree (list of points pointList, int depth)
{
    // Select axis based on depth so that axis cycles through all valid values
    var int axis := depth mod k;

    // Sort point list and choose median as pivot element
    select median by axis from pointList;

    // Create node and construct subtree
    node.location := median;
    node.leftChild := kdtree(points in pointList before median, depth+1);
    node.rightChild := kdtree(points in pointList after median, depth+1);
    return node;
}
```



Building kD-trees

- Given the following 7 points, build the kd tree ($k=2$)

$(7, 8), (4, 7), (2, 9), (7, 4), (5, 1), (3, 5), (6, 6)$

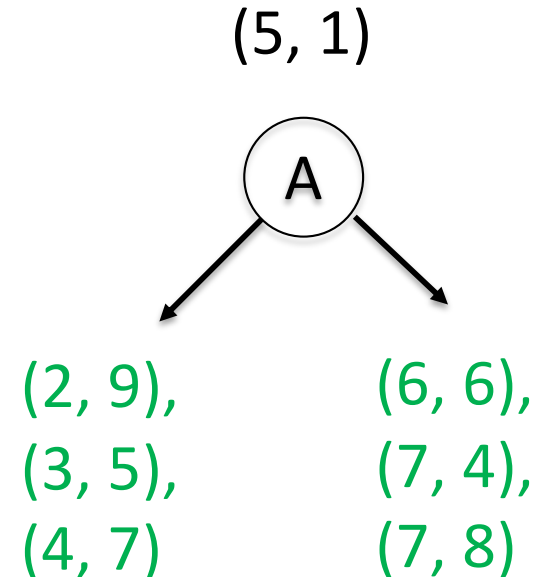
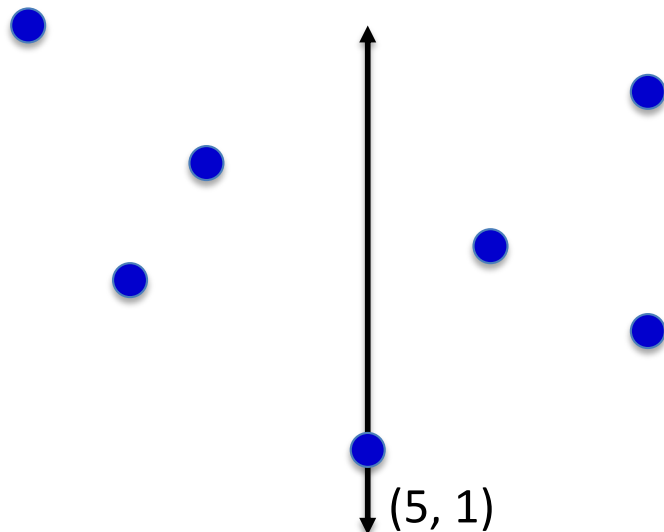
- Select an axis for splitting, let's say start with x-axis

- Find the median value of x-axis values

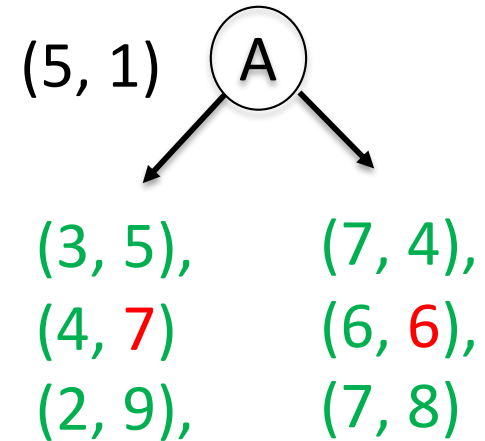
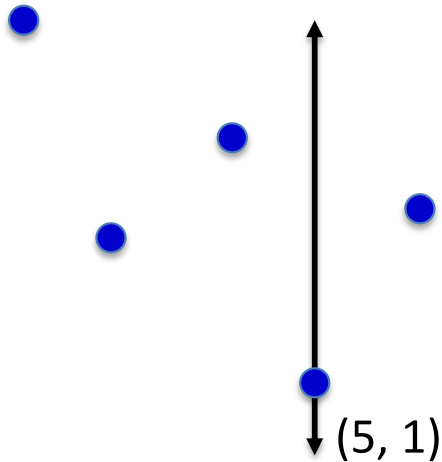
sort the points along x-axis:

$(2, 9), (3, 5), (4, 7), (5, 1), (6, 6), (7, 4), (7, 8)$

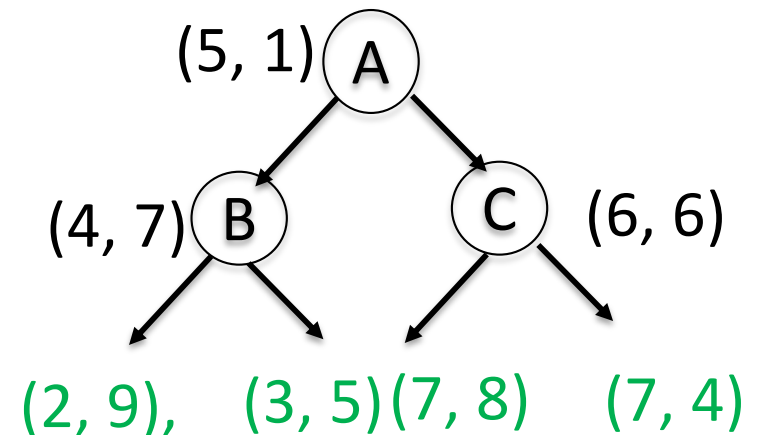
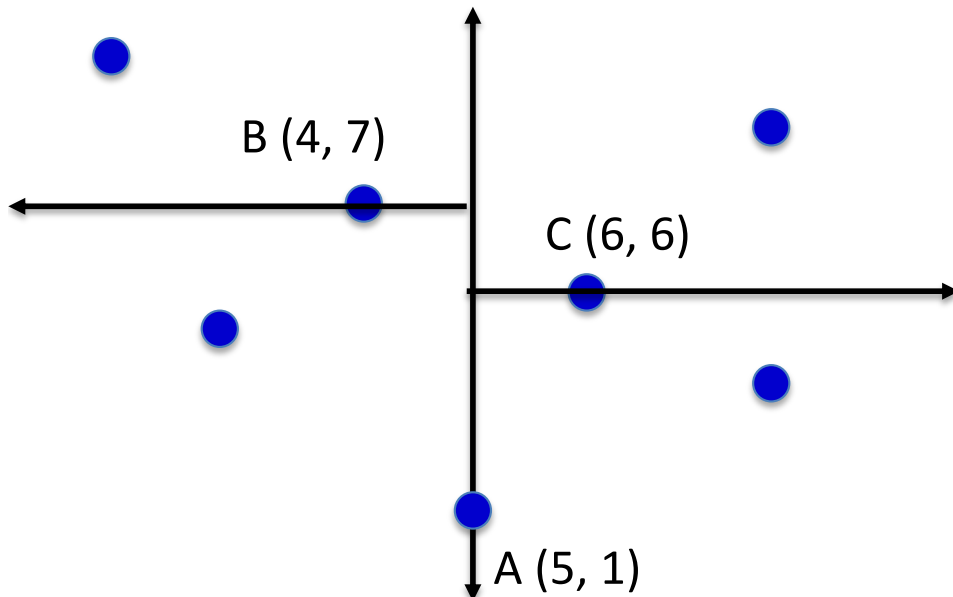
- split into two sub-trees:



Building kD-trees



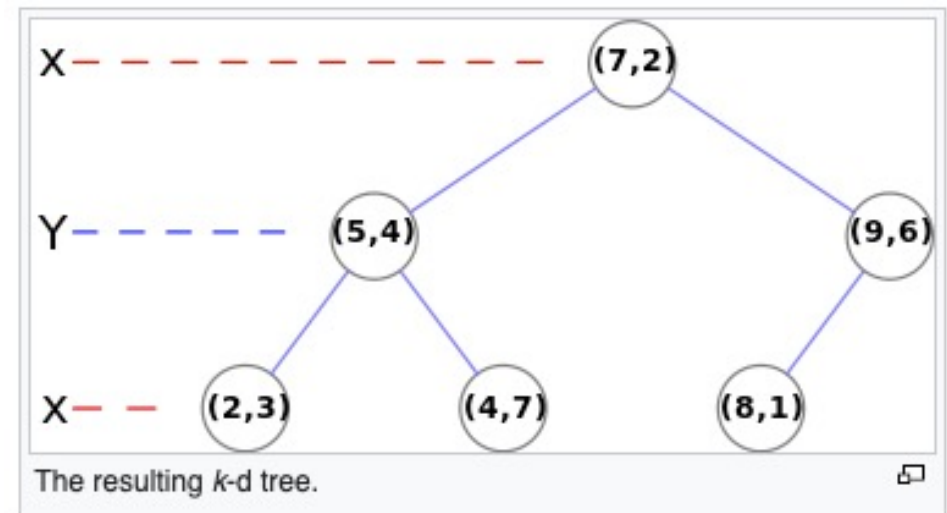
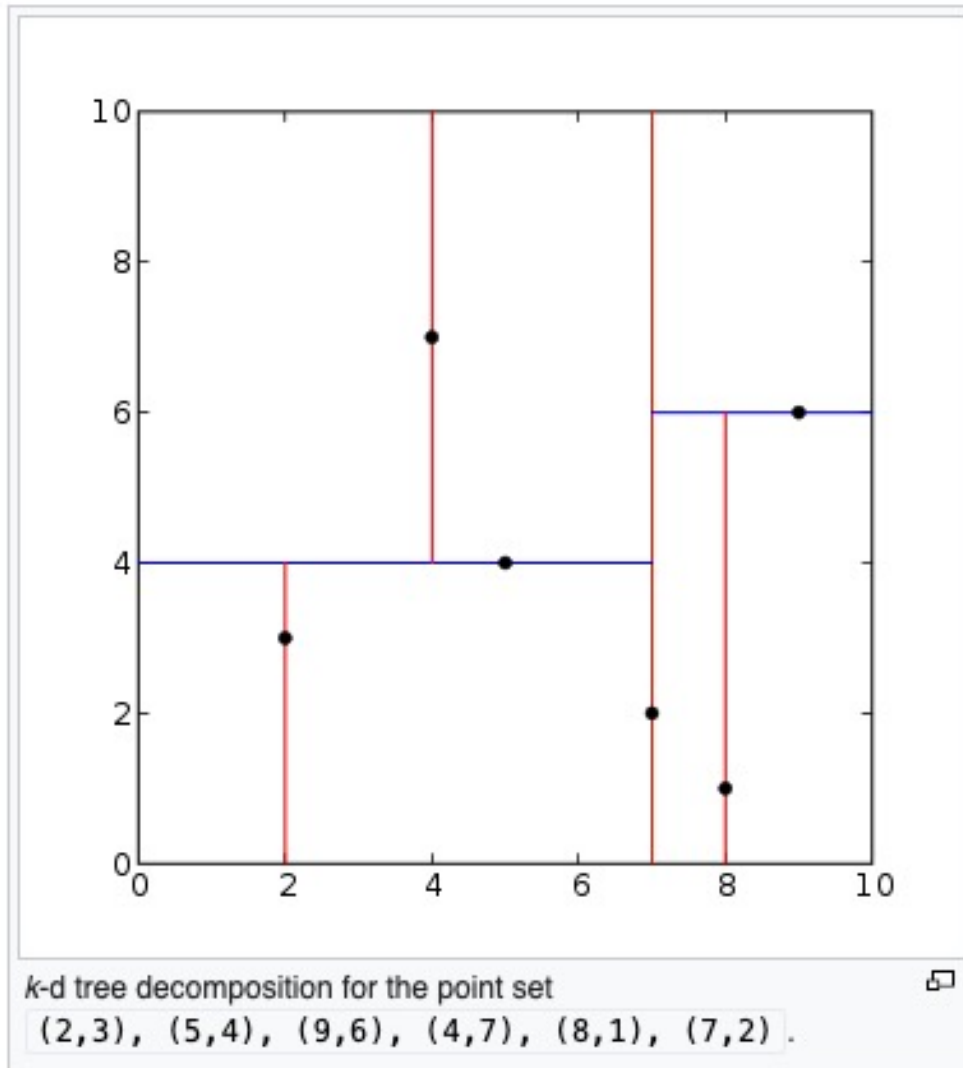
- In each sub-space, split the points along the next axis, ie., y-axis



More on *k*D-trees

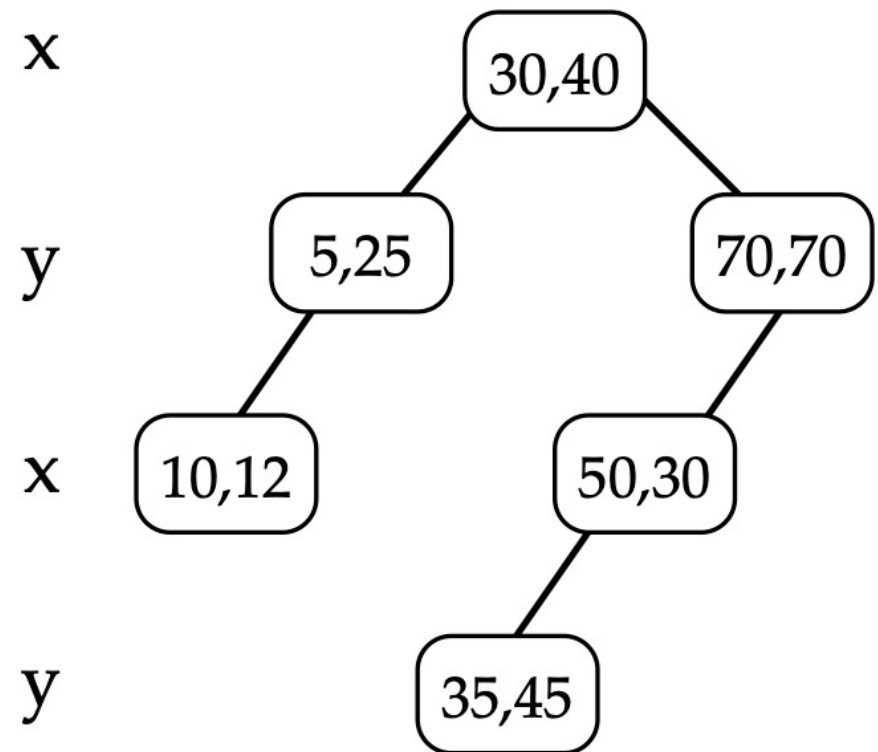
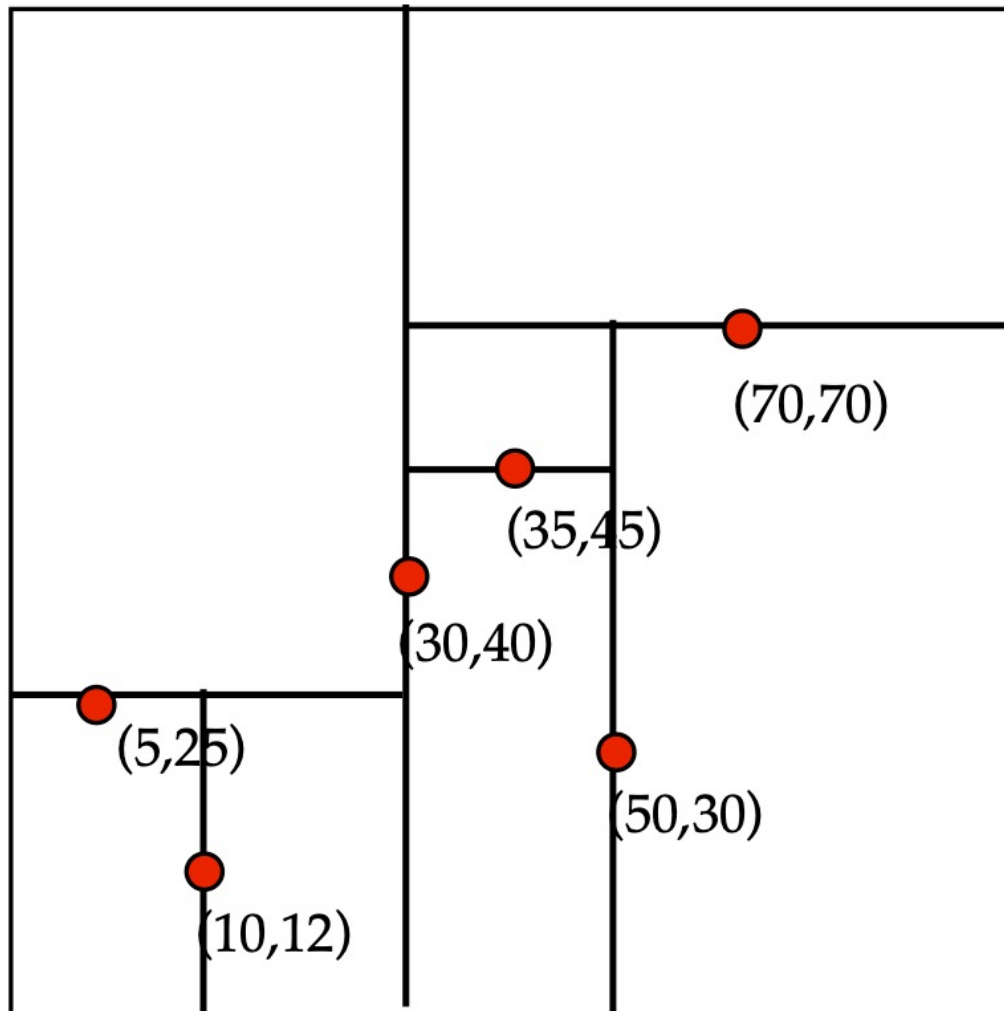
- Complexity depends on depth of tree, given by logarithm of number of nodes
- How to build a good tree? Need to find good split point and split direction
 - ◆ Split direction: direction with greatest variance
 - ◆ Split point: median value along that direction
- Using value closest to mean (rather than median) can be better if data is skewed
- Can apply this recursively

More Examples of Kd Tree

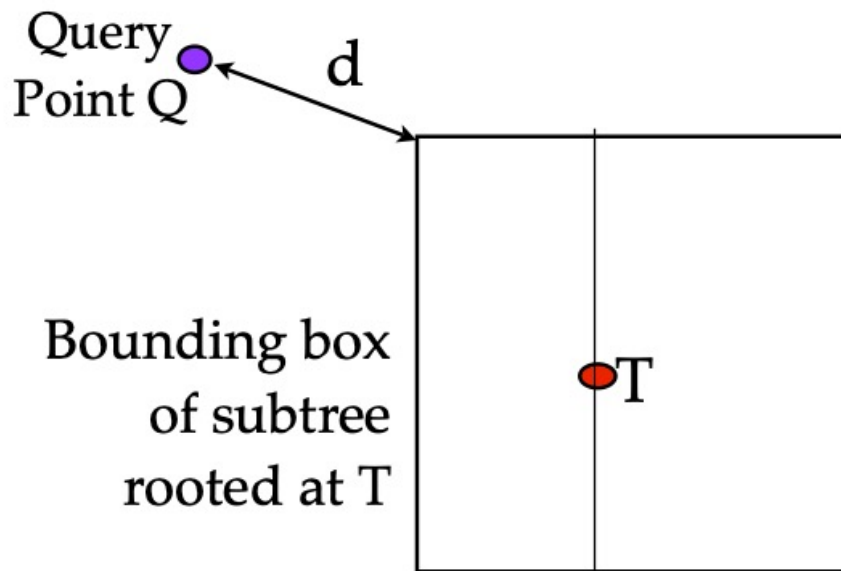


More Examples of Kd Trees

insert: (30,40), (5,25), (10,12), (70,70), (50,30), (35,45)



Using Kd Tree



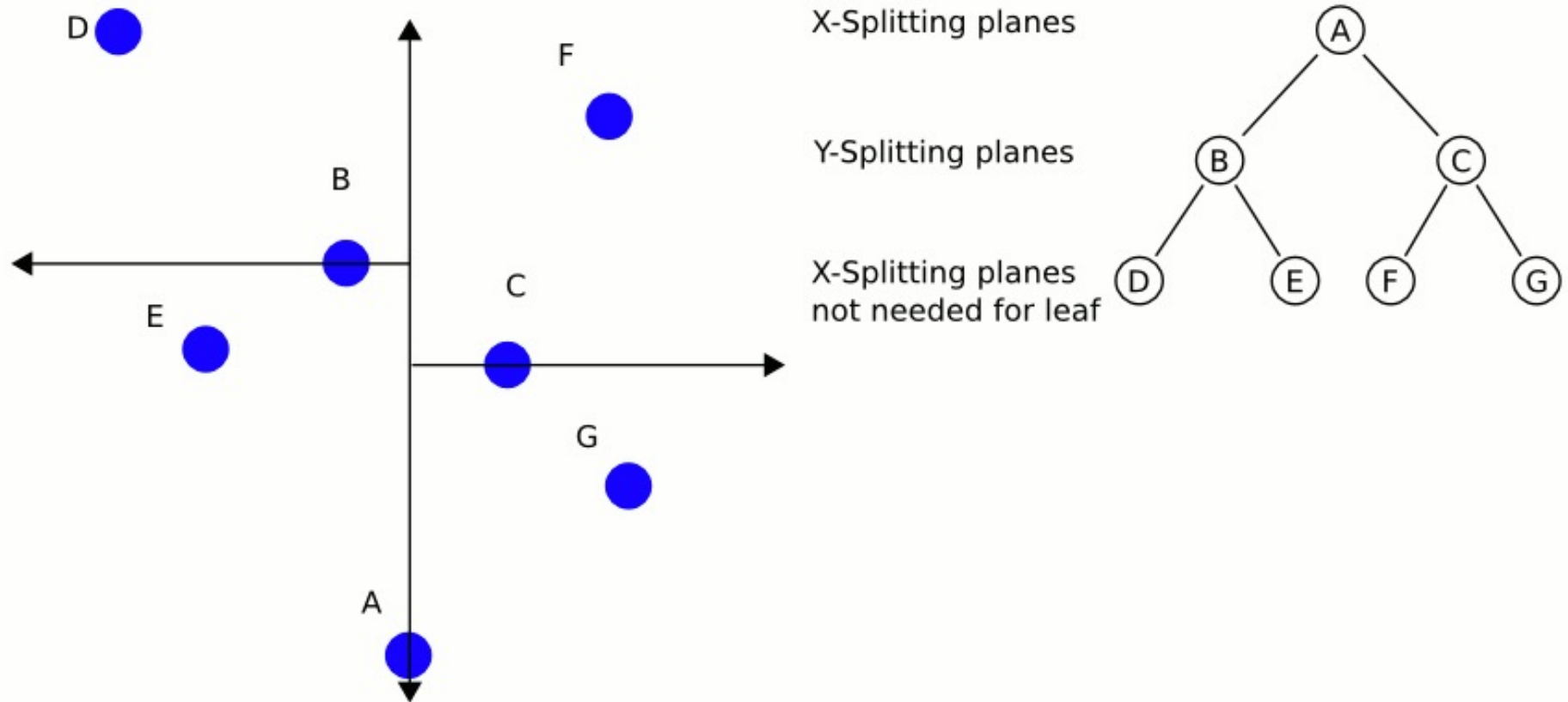
C -- current best point

If $d > \text{dist}(C, Q)$, then no point in $\text{BB}(T)$ can be closer to Q than C .
Hence, no reason to search subtree rooted at T .

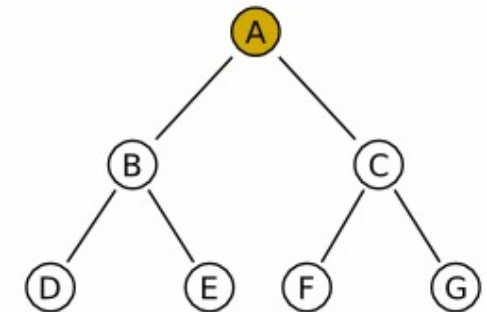
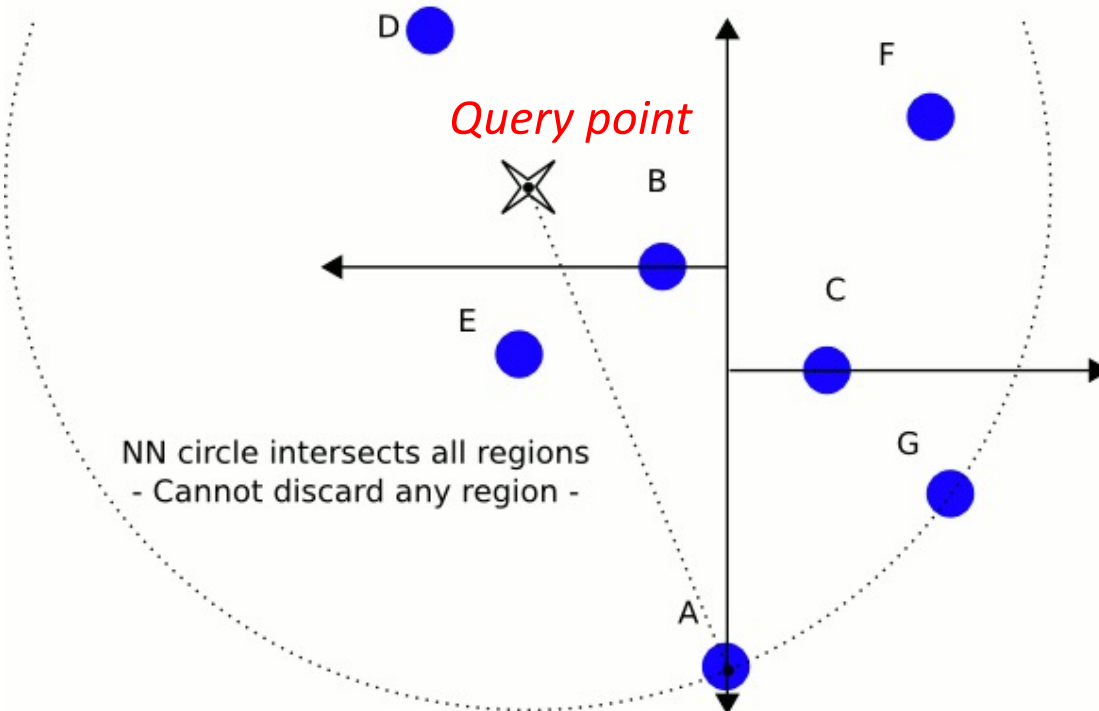
Update the best point so far, if T is better:
if $\text{dist}(C, Q) > \text{dist}(T.\text{data}, Q)$, $C := T.\text{data}$

Recurse, but start with the subtree “closer” to Q :
First search the subtree that would contain Q if we were inserting Q below T .

Using kD-trees

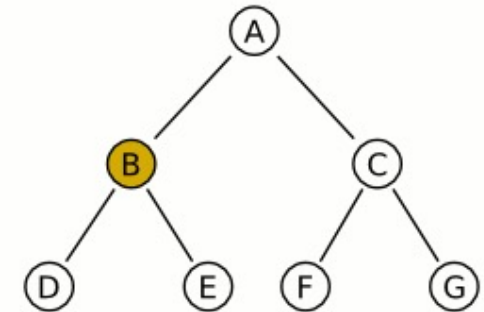
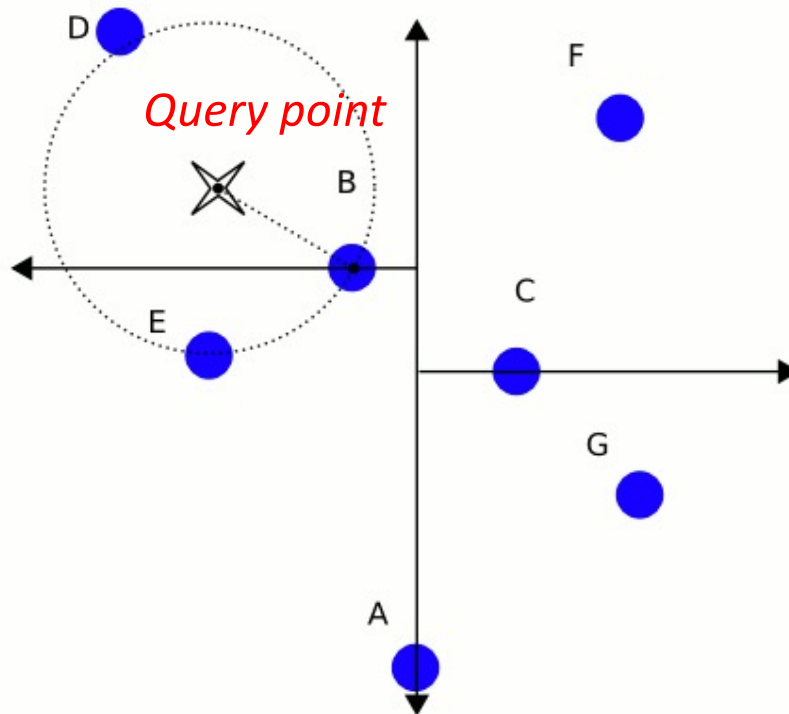


Using kD-trees



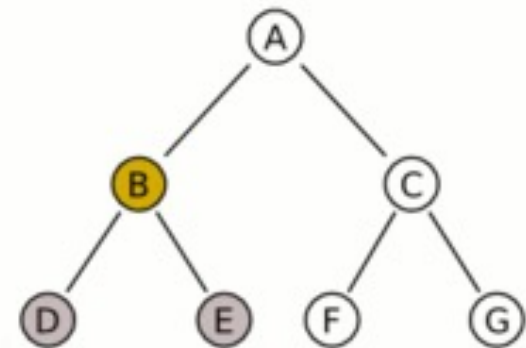
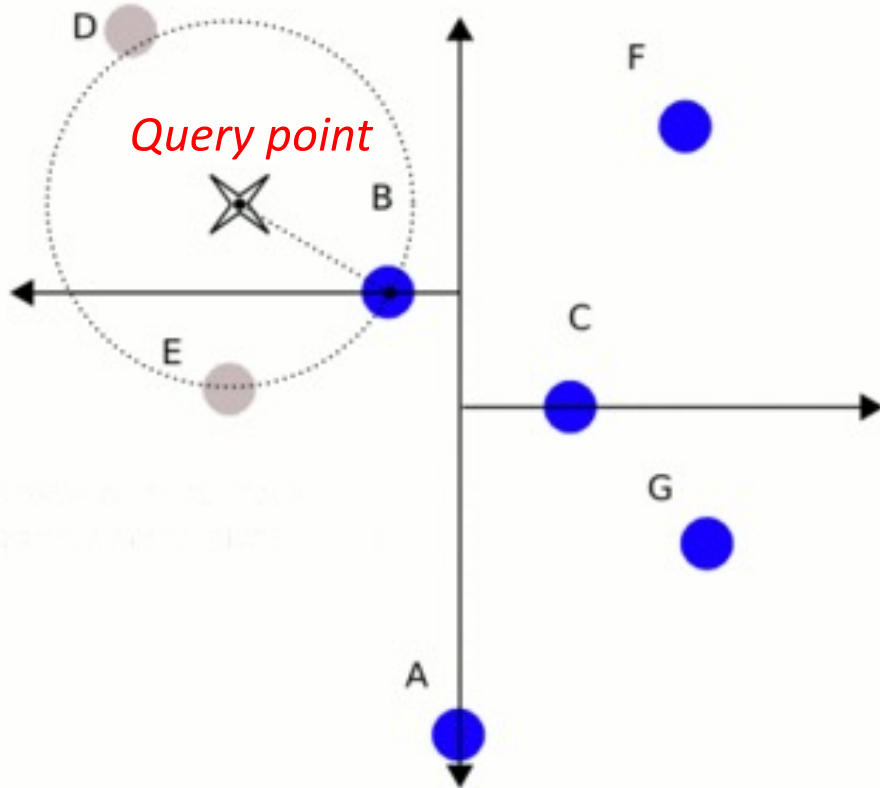
Start at A, then proceed in depth-first search (maintain a stack of parent-nodes if using a singly-linked tree). Set best estimate to A's distance. Then examine left child node

Using kD-trees



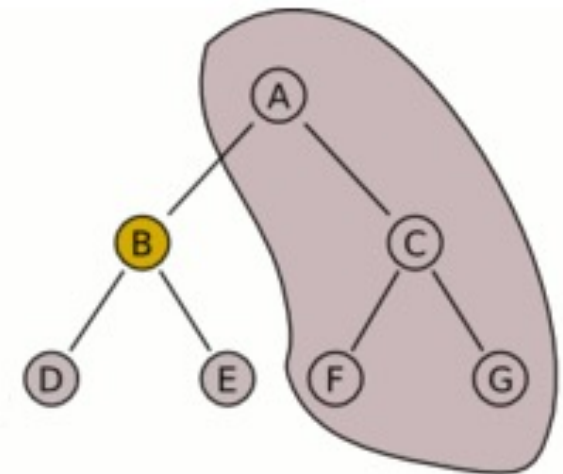
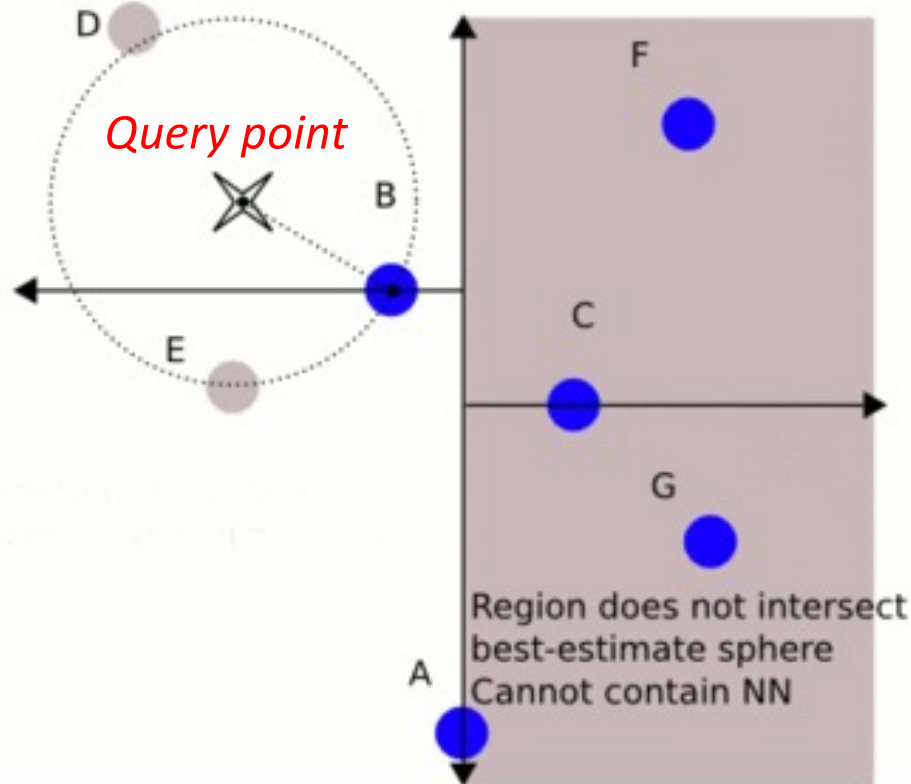
Calculate B's distance and compare against best estimate
- It is smaller distance, so update best estimate. Examine children (left then right)

Using kD-trees



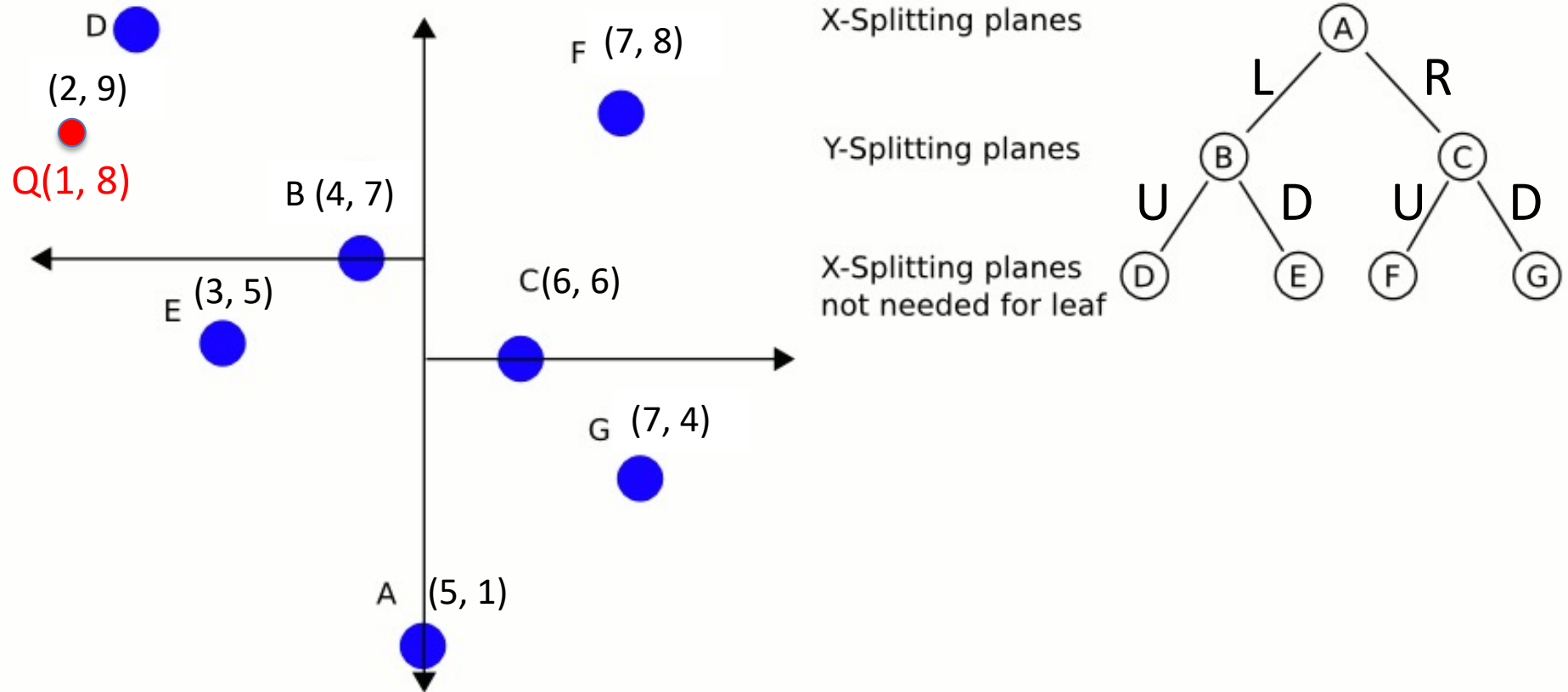
D & E Discarded as B
(already visited) is closer.
B is the best estimate for B's sub-branch
Proceed back to parent node

Using kD-trees

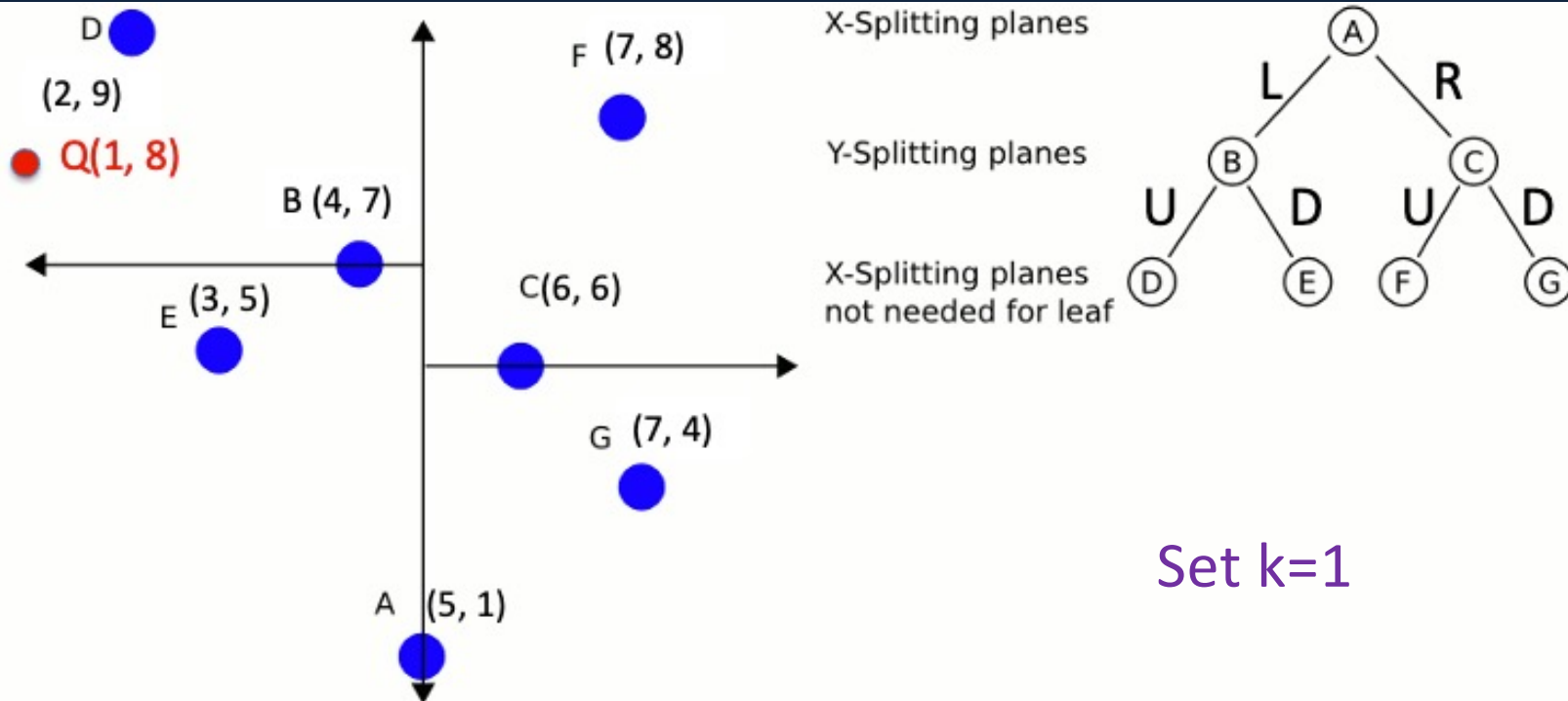


A's children have all been searched,
B is the best estimate for entire tree

Using KD Tree Example

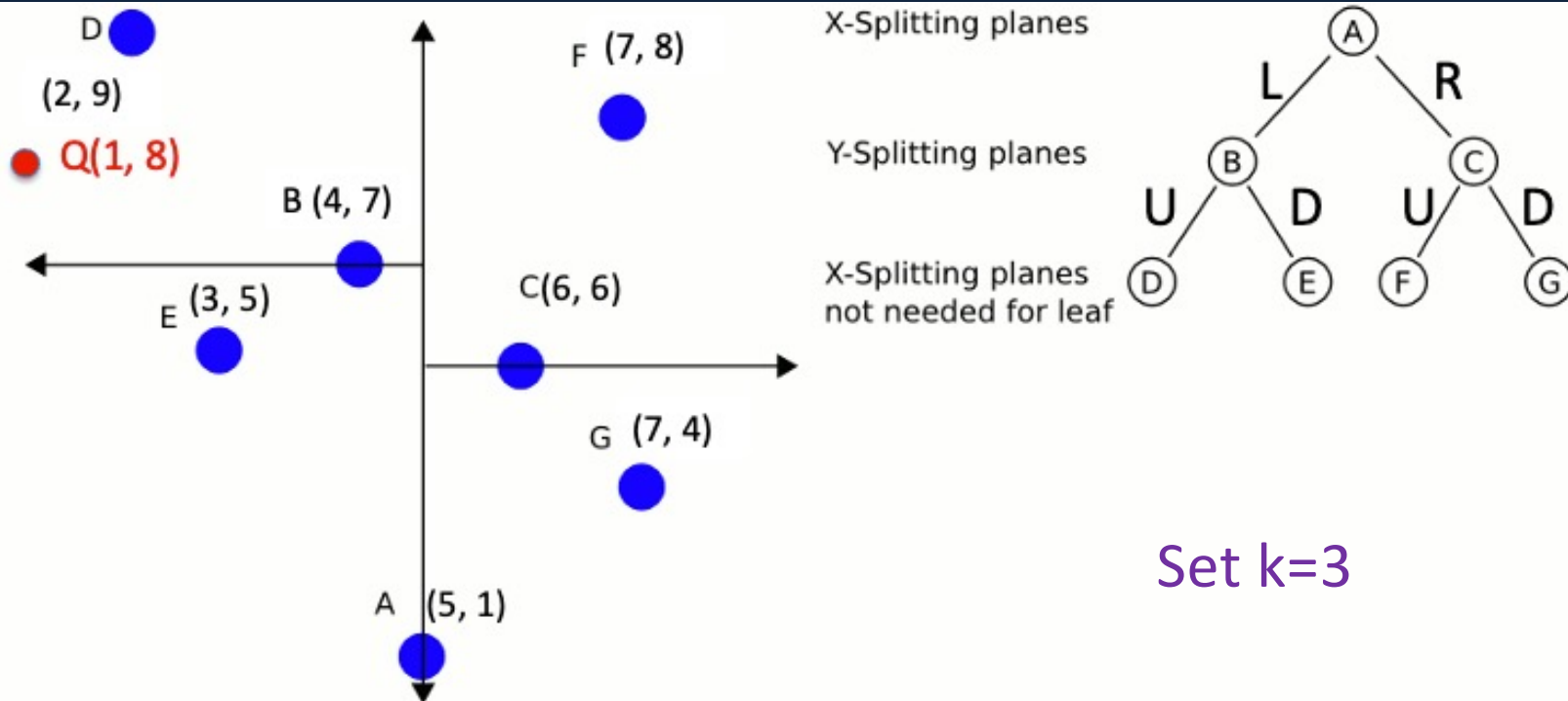


Using kD-trees



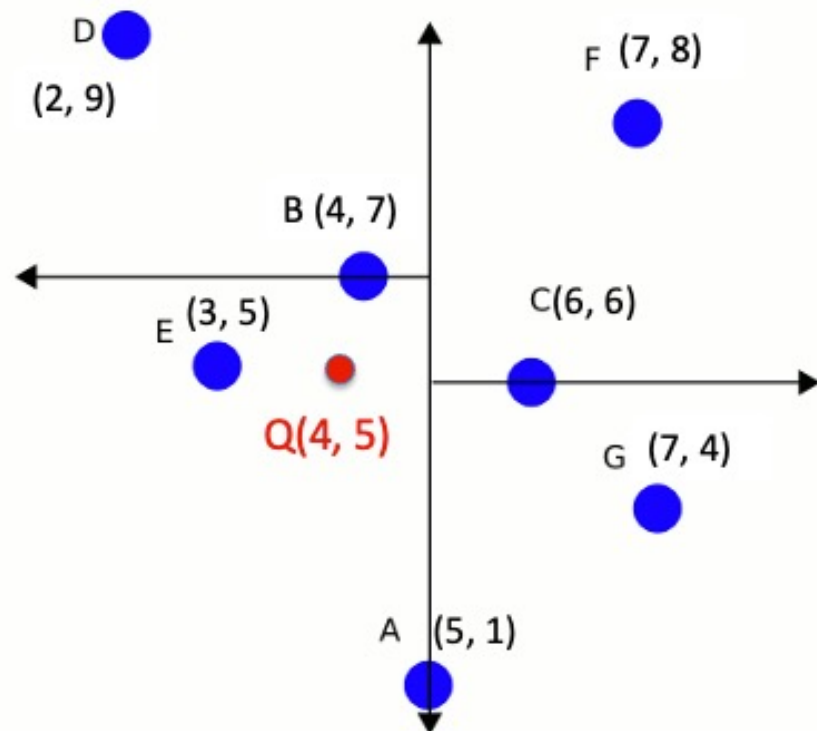
1. distance (Q, A), best estimate = 8.02 { }
2. Left, distance (Q, B)=3.16, update best estimate = 3.16 { }
3. Up, leaf node, distance (Q, D)= 1.414 , add D to top K neighbor queue {D(1.4)}
4. Backtrack, is there need to explore the lower half of the left tree? Yes, distance from x to split line is 1, less than current best 1.414 ...
5. Distance (Q, E) = 3.6, add to queue {D(1.4)}
6. Backtrack to B and then A, B and A's distance larger than D, {D(1.4)}
7. No need to explore the right side of A because the best distance is 4 > 3.6

Using kD-trees



1. distance (Q, A), best estimate = 8.02 { }
2. Left, distance (Q, B)=3.16, update best estimate = 3.16 { }
3. Up, leaf node, distance (Q, D)= 1.414 , add D to top K neighbor queue {D(1.4)}
4. Backtrack, is there need to explore the lower half of the left tree? Yes, distance from x to split line is 1, less than current best 1.414 ...
5. Distance (Q, E) = 3.6, add to queue {E(3.6), D(1.4)}
6. Backtrack, add B to queue {E(3.6), B(3.16), D(1.4)}
7. No need to explore the right side of A because the best distance is 4 > 3.6

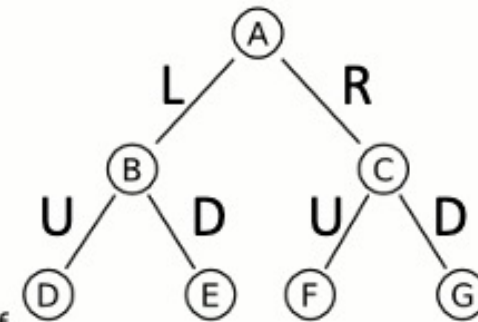
Practice Question



X-Splitting planes

Y-Splitting planes

X-Splitting planes
not needed for leaf



K=1? K=3?

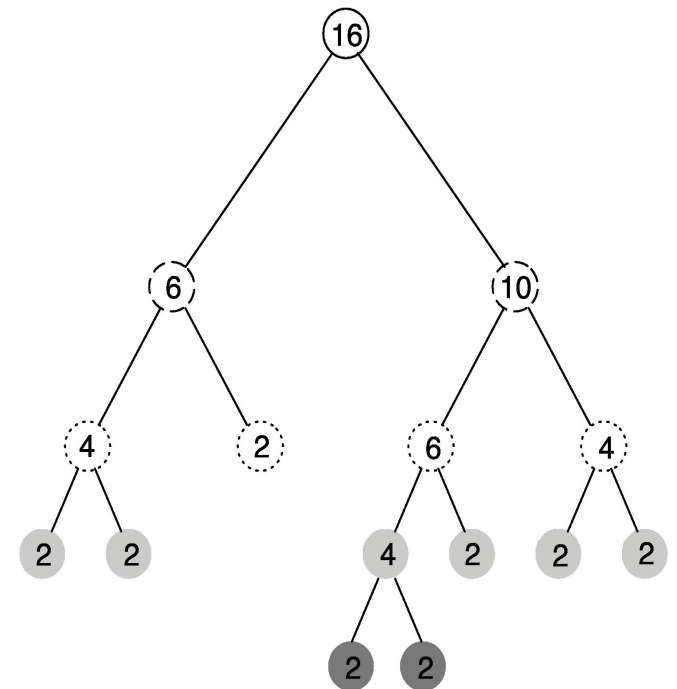
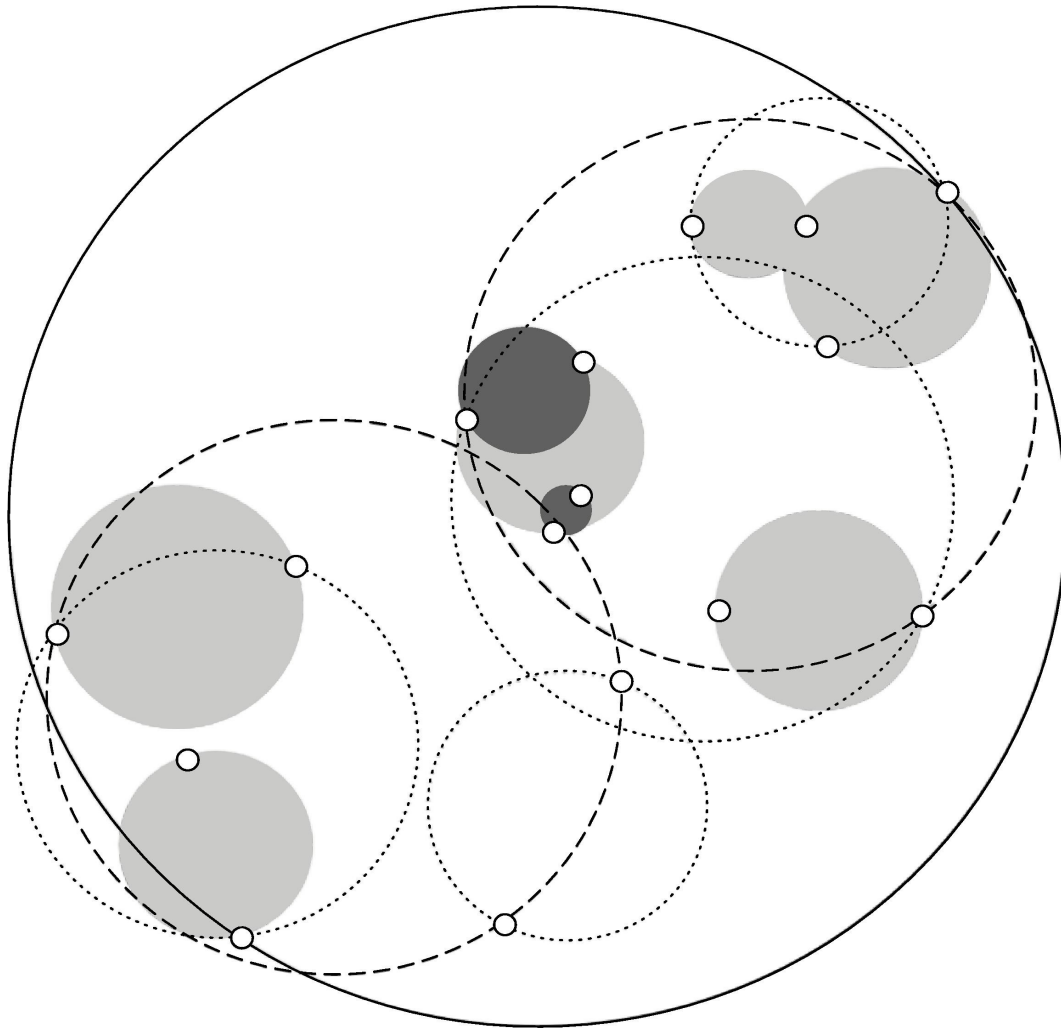
Building trees incrementally

- Big advantage of instance-based learning: classifier can be updated incrementally
 - ♦ Just add new training instance!
- Can we do the same with *kD*-trees?
- Heuristic strategy:
 - ♦ Find leaf node containing new instance
 - ♦ Place instance into leaf if leaf is empty
 - ♦ Otherwise, split leaf according to the longest dimension (to preserve squareness)
- Tree should be re-built occasionally (i.e. if depth grows to twice the optimum depth)

Ball trees

- Observation: no need to make sure that regions don't overlap
- Can use balls (hyperspheres) instead of hyperrectangles
 - ♦ *A ball tree* organizes the data into a tree of k -dimensional hyperspheres
 - ♦ Normally allows for a better fit to the data and thus more efficient search

Ball tree example



Building ball trees

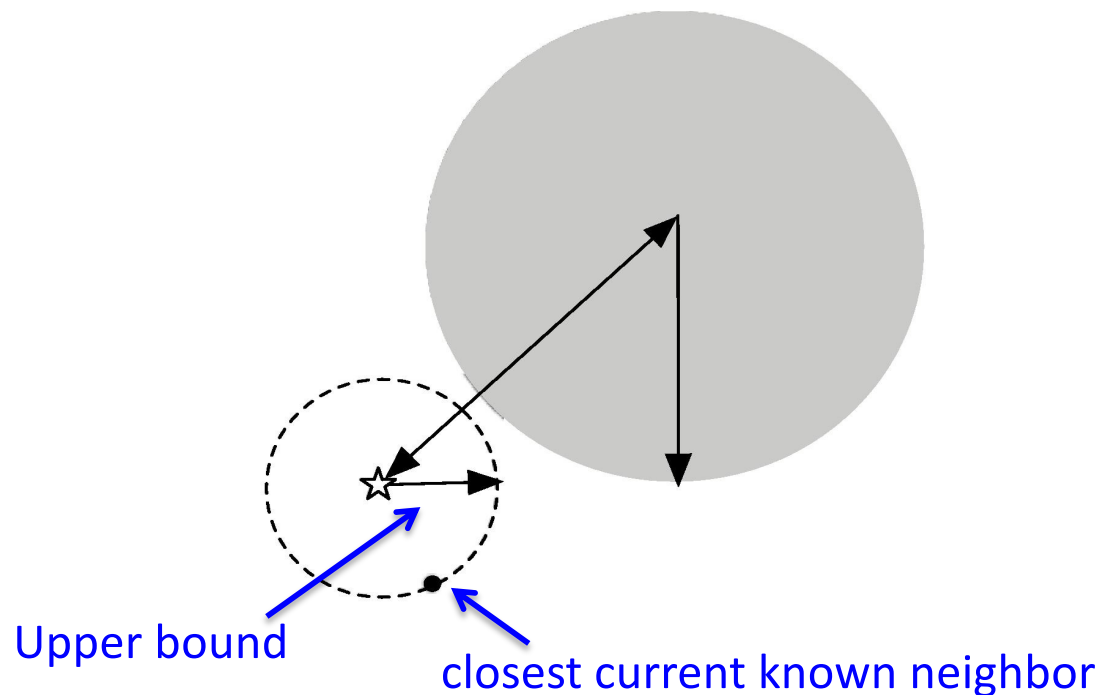
- Ball trees are built top down (like *kD*-trees)
- Basic problem: splitting a ball into two

```
function construct_balltree is
  input:
    D, an array of data points
  output:
    B, the root of a constructed ball tree
  if a single point remains then
    create a leaf B containing the single point in D
    return B
  else
    let c be the dimension of greatest spread
    let L,R be the sets of points lying to the left and right of the median along dimension c
    create B with two children:
      B.pivot = c
      B.child1 = construct_balltree(L),
      B.child2 = construct_balltree(R)
    return B
  end if
end function
```

Using ball trees

- Nearest-neighbor search is done using the same backtracking strategy as in *kD*-trees
- Ball can be ruled out from consideration if: distance from target to ball's center exceeds ball's radius plus current upper bound.

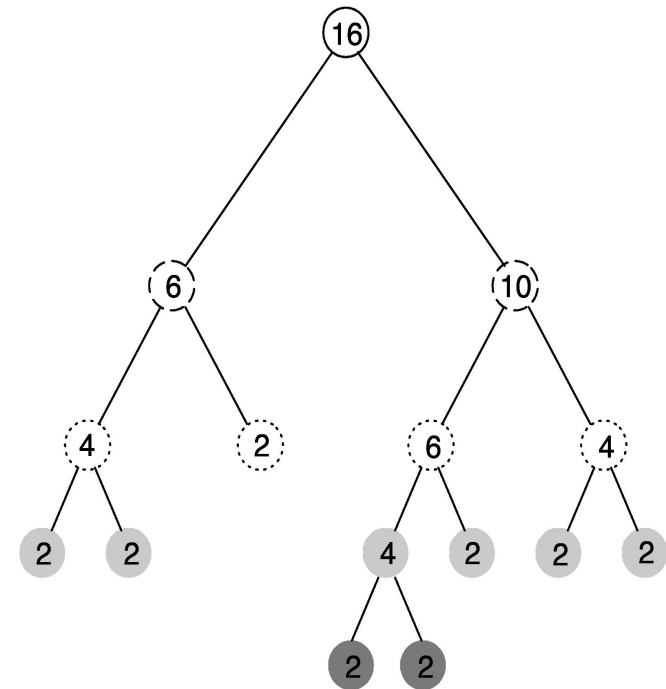
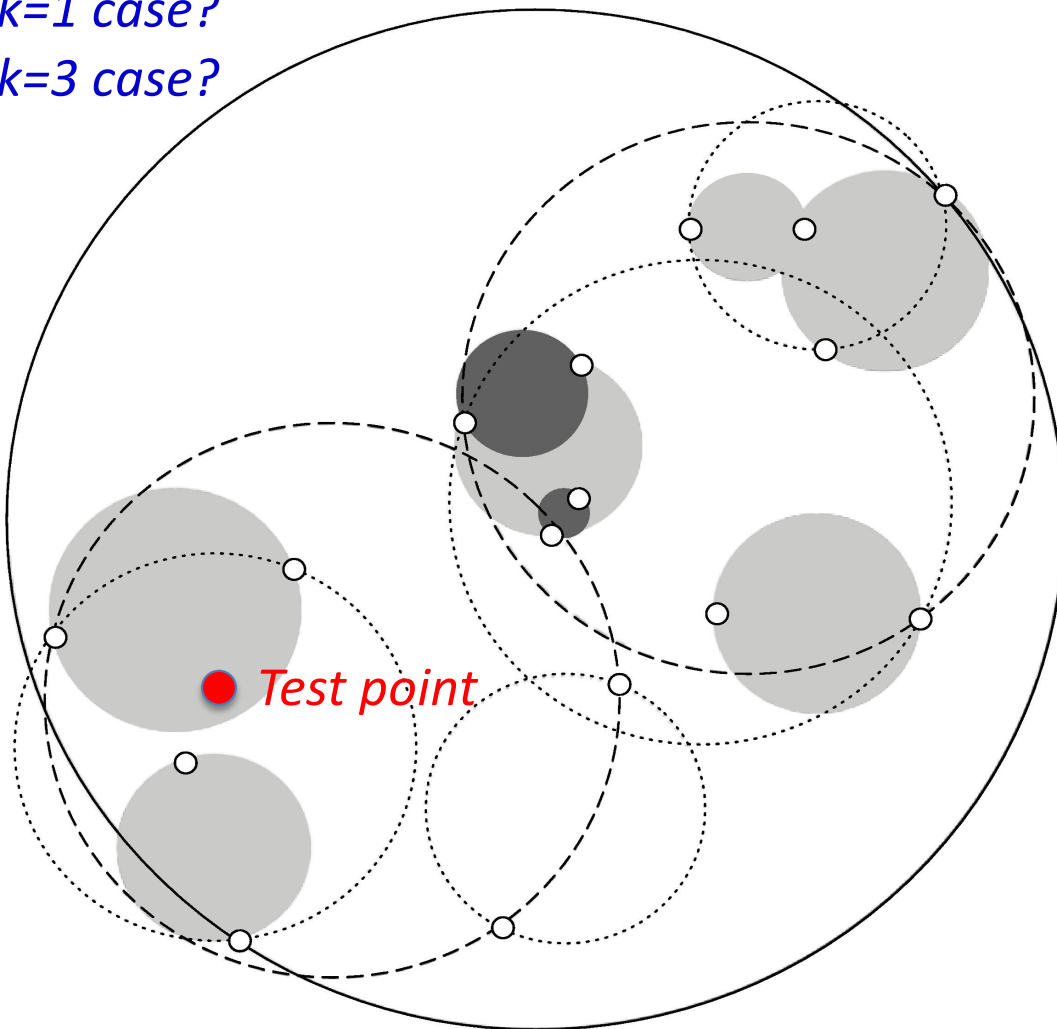
(two circles no overlap)



Ball tree example

k=1 case?

k=3 case?

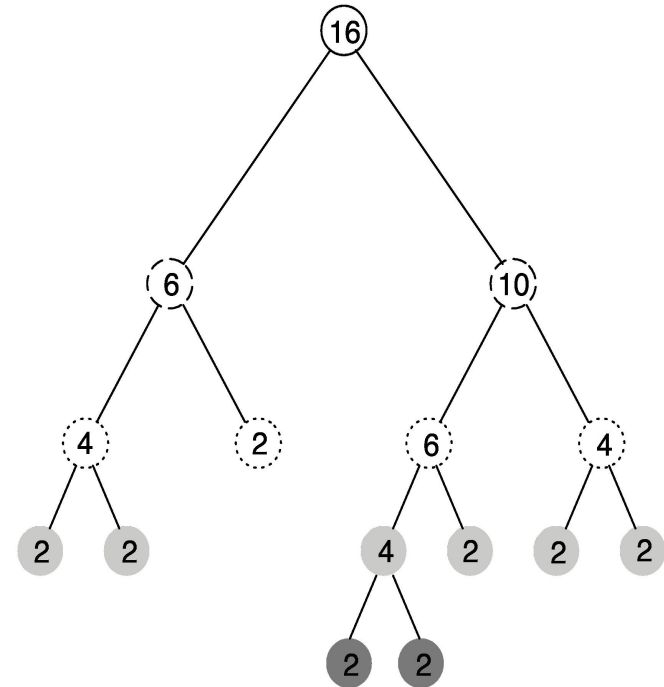
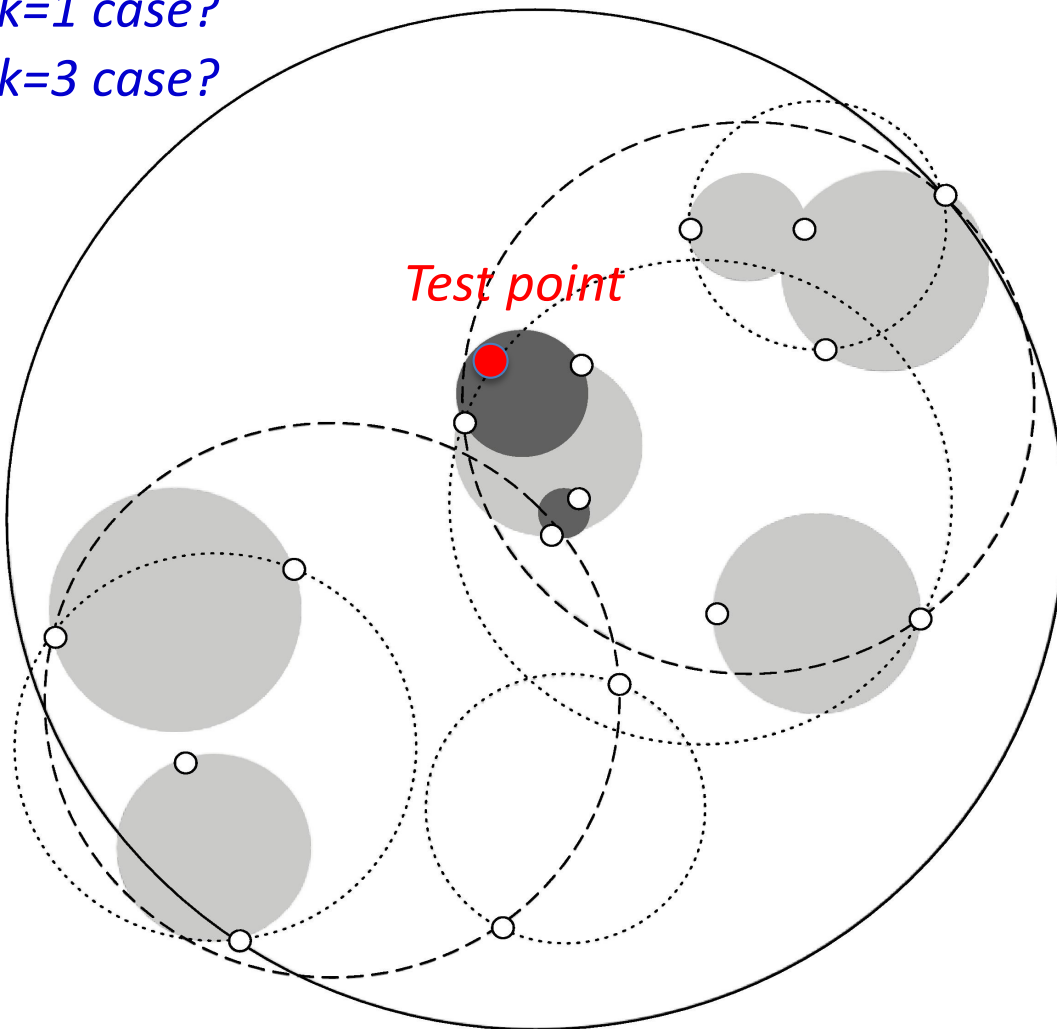


Ball can be ruled out from consideration if: distance from target to ball's center exceeds ball's radius plus current upper bound

Ball tree example

k=1 case?

k=3 case?



Ball can be ruled out from consideration if: distance from target to ball's center exceeds ball's radius plus current upper bound

Nearest Neighbor with Ball Tree

- At each node B , it may perform one of three operations, before finally returning an updated version of the priority queue:
 - If the distance from the test point t to the current node B is greater than radius + the furthest point in Q , ignore B and return Q .
 - If B is a leaf node, scan through every point enumerated in B and update the nearest-neighbor queue appropriately. Return the updated queue.
 - If B is an internal node, call the algorithm recursively on B 's two children, searching the child whose center is closer to t first. Return the queue after each of these calls has updated it in turn.

KNN Search Algorithm

```
function knn_search is
  input:
    t, the target point for the query
    k, the number of nearest neighbors of t to search for
    Q, max-first priority queue containing at most k points
    B, a node, or ball, in the tree
  output:
    Q, containing the k nearest neighbors from within B
  if distance(t, B.pivot) ≥ distance(t, Q.first) then
    return Q unchanged
  else if B is a leaf node then
    for each point p in B do
      if distance(t, p) < distance(t, Q.first) then
        add p to Q
        if size(Q) > k then
          remove the furthest neighbor from Q
        end if
      end if
    repeat
  else
    let child1 be the child node closest to t
    let child2 be the child node furthest from t
    knn_search(t, k, Q, child1)
    knn_search(t, k, Q, child2)
  end if
end function[2]
```

Discussion of nearest-neighbor learning

- Scikit implementation: <http://scikit-learn.org/stable/modules/neighbors.html>
- Often very accurate
- Assumes all attributes are equally important
 - Remedy: attribute selection or weights
- Possible remedies against noisy instances:
 - Take a majority vote over the k nearest neighbors
 - Removing noisy instances from dataset (difficult!)
- Statisticians have used k -NN since early 1950s
 - If $n \rightarrow \infty$ and $k/n \rightarrow 0$, error approaches minimum
- k D-trees become inefficient when number of attributes is too large (approximately > 10)
- Ball trees (which are instances of *metric trees*) work well in higher-dimensional spaces