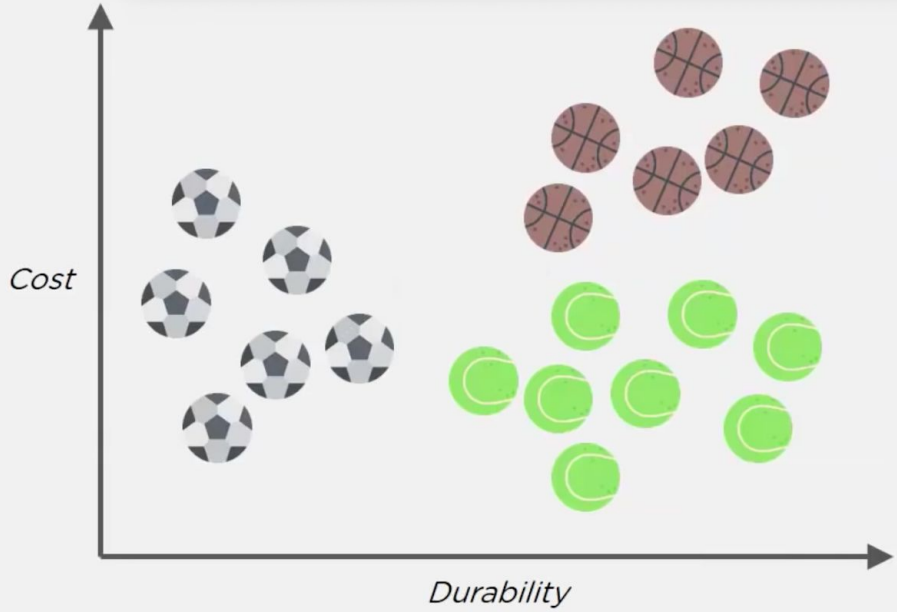
Probablistic Learning

K Nearest Neighbours

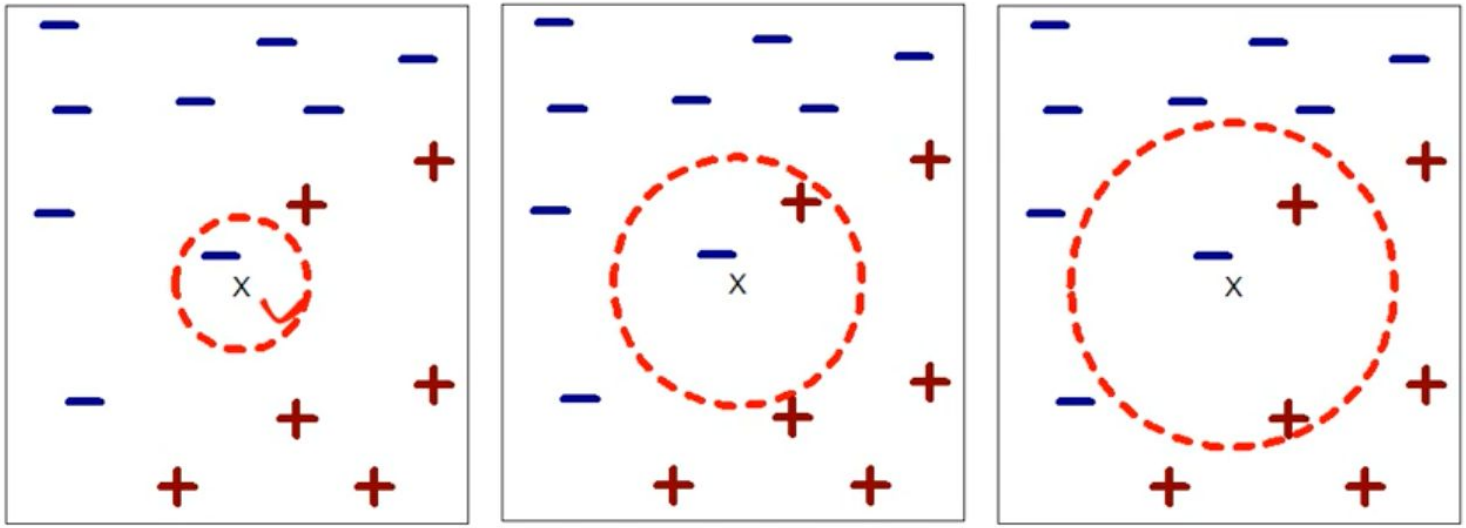
Explain K Nearest Neighbor algorithm.



Consider there are 3 clusters:

- Football
- Basketball
- Tennis Ball

Definition of Nearest Neighbor



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

K-nearest neighbors of a record x are data points that have the k smallest distance to x

PROXIMITY METRICS

> EUCLIDEAN DISTANCE,

> HAMMING DISTANCE,

> MANHATTAN DISTANCE (CITY BLOCK)

> MINKOWSKY

> CHEBYCHEV DISTANCE

Some frequently used distance functions.

Canberra :

$$d(x,y) = \sum_{i=1}^{m} \frac{|x_i - y_i|}{|x_i + y_i|} \quad (2)$$

Minkowsky :

$$d(x,y) = \left(\sum_{i=1}^{m} |x_i - y_i|^r \right)^{1/r} \quad (3)$$

Chebychev :

$$d(x,y) = \max_{i=1}^{m} |x_i - y_i| \quad (4)$$

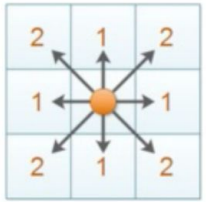
Euclidean :

$$d(x,y) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2} \quad (5)$$

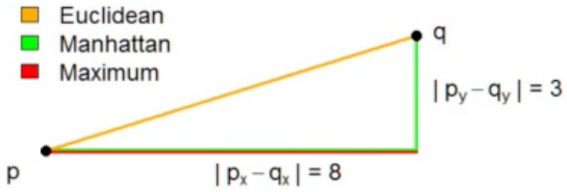
Manhattan / city - block :

$$d(x,y) = \sum_{i=1}^{n} |x_i - y_i| \quad (6)$$

Manhattan Distance



$$|x_1 - x_2| + |y_1 - y_2|$$



K-NN Classification

Example

Name	Acid Durability	Strength	Class
Type-1	7	7	Bad
Type-2	7	4	Bad
Type-3	3	4	Good
Type-4	1	4	Good

Test-Data ➔ acid durability=3, and strength=7, class=?

Similarity

- Calculated using distance measure like Euclidean

$$\mathrm{d}(\mathbf{p}, \mathbf{q}) = \mathrm{d}(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

Name	Acid Durability	Strength	Class	Distance
Type-1	7	7	Bad	Sqrt((7-3)^2+(7-7)^2)=4
Type-2	7	4	Bad	5
Type-3	3	4	Good	3
Type-4	1	4	Good	3.6

Rank these attributes

Name	Acid Durability	Strength	Class	Distance	Rank
Type-1	7	7	Bad	4	3
Type-2	7	4	Bad	5	4
Type-3	3	4	Good	3	1
Type-4	1	4	Good	3.6	2

k=1

Name	Acid Durability	Strength	Class	Distance	Rank
Type-1	7	7	Bad	4	3
Type-2	7	4	Bad	5	4
Type-3	3	4	Good	3	1
Type-4	1	4	Good	3.6	2

Based on immediate neighbour, Good

k=2

Name	Acid Durability	Strength	Class	Distance	Rank
Type-1	7	7	Bad	4	3
Type-2	7	4	Bad	5	4
Type-3	3	4	Good	3	1
Type-4	1	4	Good	3.6	2

Based on two neighbours, Good

k=3

Name	Acid Durability	Strength	Class	Distance	Rank
Type-1	7	7	Bad	4	3
Type-2	7	4	Bad	5	4
Type-3	3	4	Good	3	1
Type-4	1	4	Good	3.6	2

Based on three neighbours, 2 Goods and 1 bad, majority → Good

Nearest Neighbor Classification

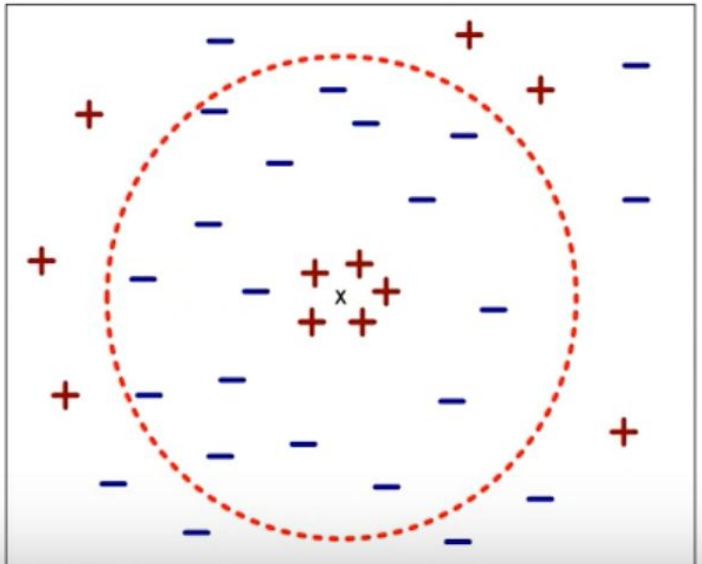
- Compute distance between two points:
  - Euclidean distance

$$d(p,q) = \sqrt{\sum_i (p_i - q_i)^2}$$

- Determine the class from nearest neighbor list
  - take the majority vote of class labels among the k-nearest neighbors
  - Weigh the vote according to distance
    - weight factor, $w = 1/d^2$

Nearest Neighbor Classification…

- Choosing the value of k:
  - If k is too small, sensitive to noise points
  - If k is too large, neighborhood may include points from other classes



So, What just happened? What exactly is K-NN?

- Simplest classification technique when there is no prior knowledge about the distribution of data.
- Computation is delayed until classification.
- 'K' in K-NN refers to the number of nearest neighbours.
- K-NN is a Supervised Learning technique.
- Prediction of the type of a data point is based on the majority type of its neighbours.
- K-NN is usually associated for Classification Problem, but can be applied for Regression through kernel smoothers. (given in later slides)
- K-NN is a Lazy Learning technique. It does not 'build' models explicitly.
- K-NN is 'non-parametric' as it does not try to adhere to a mathematical model unlike linear regression.

Eager vs Lazy learners

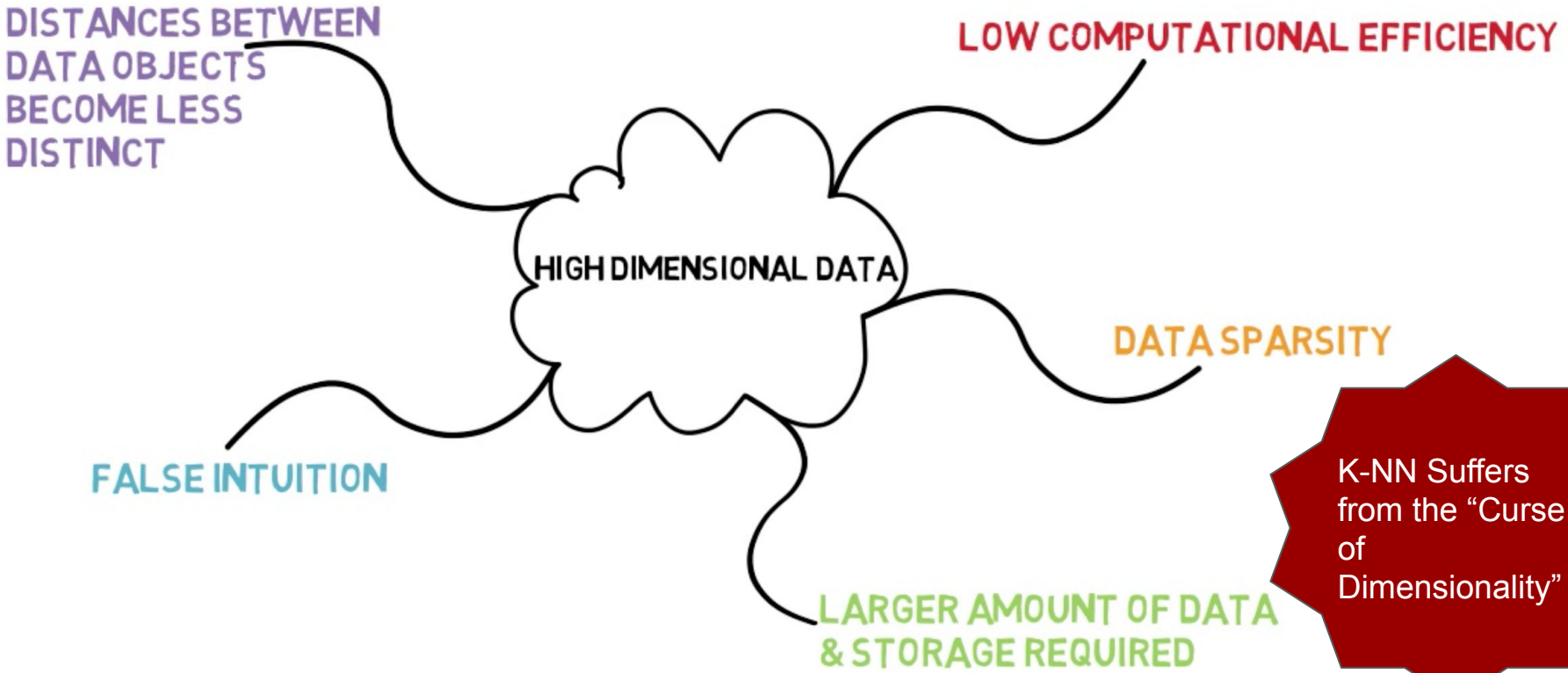
- Eager learner
  - Generalized model from training data set is constructed
  - Using the model the class of test data set is predicted
  - Example – decision tree
- Lazy learner
  - Training dataset is stored
  - On querying similarity between test data and training set records is calculated to predict the class of test data
  - Example – k-Nearest Neighbour

ADVANTAGES OF KNN

- ROBUST TO NOISY TRAINING DATA
- EFFECTIVE IS TRAINING DATA IS LARGE
- NO TRAINING PHASE

- LEARNS COMPLEX MODELS EASILY

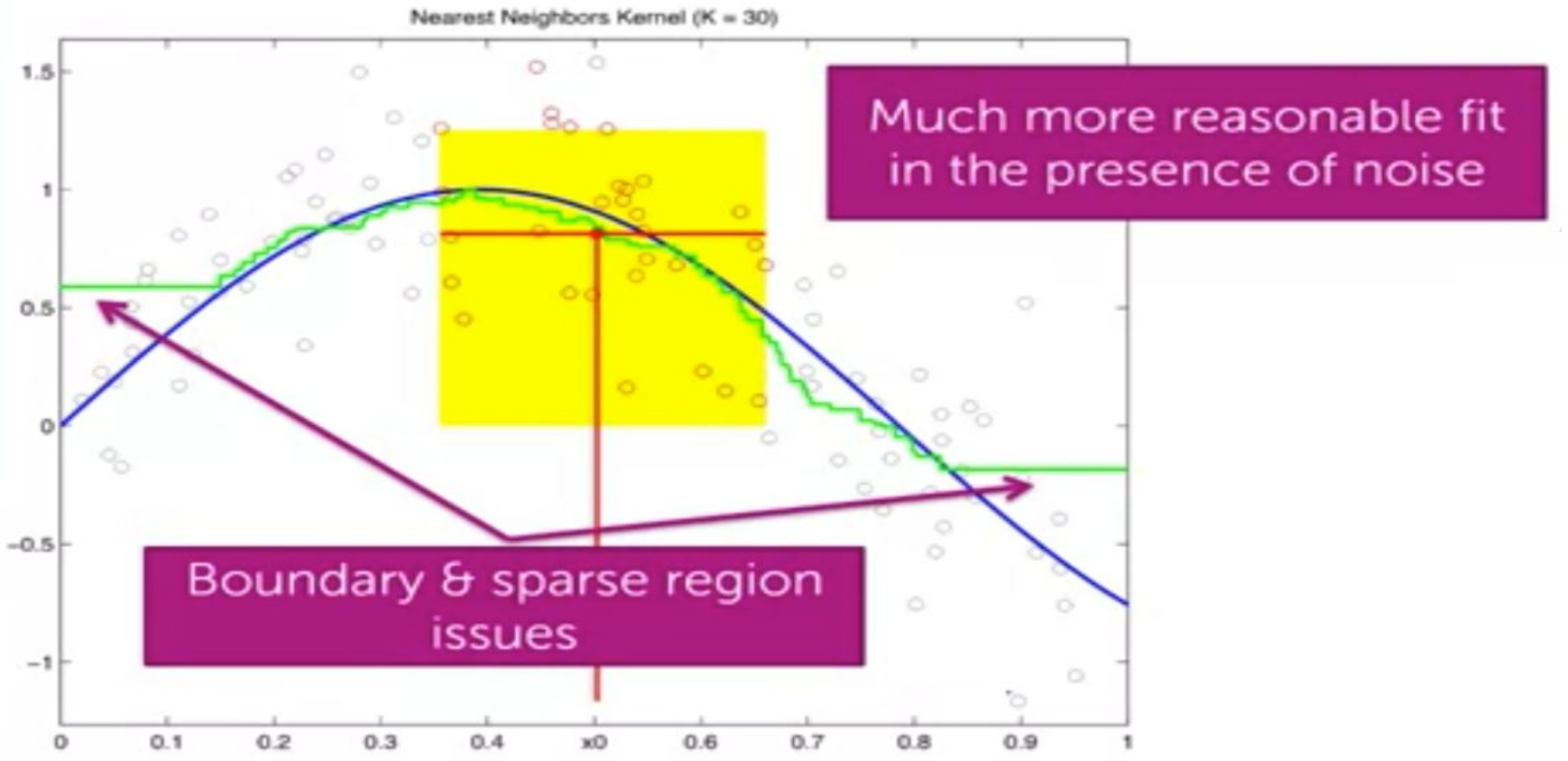
> NEED TO DETERMINE VALUE OF THE PARAMETER K



> NOT CLEAR WHICH TYPE OF DISTANCE METRIC TO USE

> COMPUTATION COST IS HIGH. K - DIMENSION TREE MAY EASE COMPUTATIONS

k-NN in practice



Issues with discontinuities

Overall predictive accuracy might be okay, but...

For example, in housing application:

- If you are a buyer or seller, this matters
- Can be a jump in estimated value of house going just from 2640 sq.ft. to 2641 sq.ft.
- Don't really believe this type of fit

Making kNN fast

- Training: O(1), but testing: O(nd)
- Reduce d: dimensionality reduction
  - simple feature selection, other methods O(d^3)

There are ‘d’ squarings and ‘d’ subtractions for d’ dimensional data.

Nearest Neighbor Classification…

- Scaling issues
  - Attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes
  - Example:
    - height of a person may vary from 1.5m to 1.8m
    - weight of a person may vary from 90lb to 300lb
    - income of a person may vary from $10K to $1M

SOLVE....

Apply K-NN Algorithm and predict the type of fruit or food type to which 'Tomato' (Sweet = 6, Crunch = 4) belongs to.

When 'K' is not given assume k=1

Ingredient	Sweet	Crunch	Food Type
Grape	8	5	Fruit
Green Peas	3	7	Vegetable
Nuts	3	6	Protein
Orange	7	3	Fruit

Ingredient	Sweet	chrunch	Food Type
Grape	8	5	FRUIT
Greenbean	3	7	Vegetable
Nuts	3	6	Protein
Orange	7	3	FRUIT

Euclidean distance.

$D(\text{Tomato, Grape}) = \sqrt{(6-8)^2+(4-5)^2} = \sqrt{5} = 2.2$

$D(\text{Tomato, Greenbean}) = \sqrt{(6-3)^2+(4-7)^2} = \sqrt{18} = 4.2$

$D(\text{Tomato, Nuts}) = \sqrt{(6-3)^2+(4-6)^2} = \sqrt{13} = 3.6$

$D(\text{Tomato, Orange}) = \sqrt{(6-7)^2+(4-3)^2} = \sqrt{2} = 1.4$

Minimum

Since distance of Tomato from Orange is minimum.

∴ Tomato will belong to FRUIT ANS..

SOLVE....

Apply K-NN Algorithm and predict the Class for 'X' (P1 = 3, P2 = 7). K = 3

P_1	P_2	Class
7	7	False
7	4	False
3	4	True
1	4	True

	P_1	P_2	Class
i)	7	7	False
ii)	7	4	False
iii)	3	4	True
iv)	1	4	True

$X(P_1 = 3, P_2 = 7)$ will belong to class (TRUE)

ANS..

Euclidean distance $= \sqrt{(X_H - H_1)^2 + (X_W - W)^2 + \dots}$

X_H → Observed Value

H_1 → Actual Value

$D(X, i) = \sqrt{(3-7)^2 + (7-7)^2} = 4 \longrightarrow N3 \rightarrow FALSE$

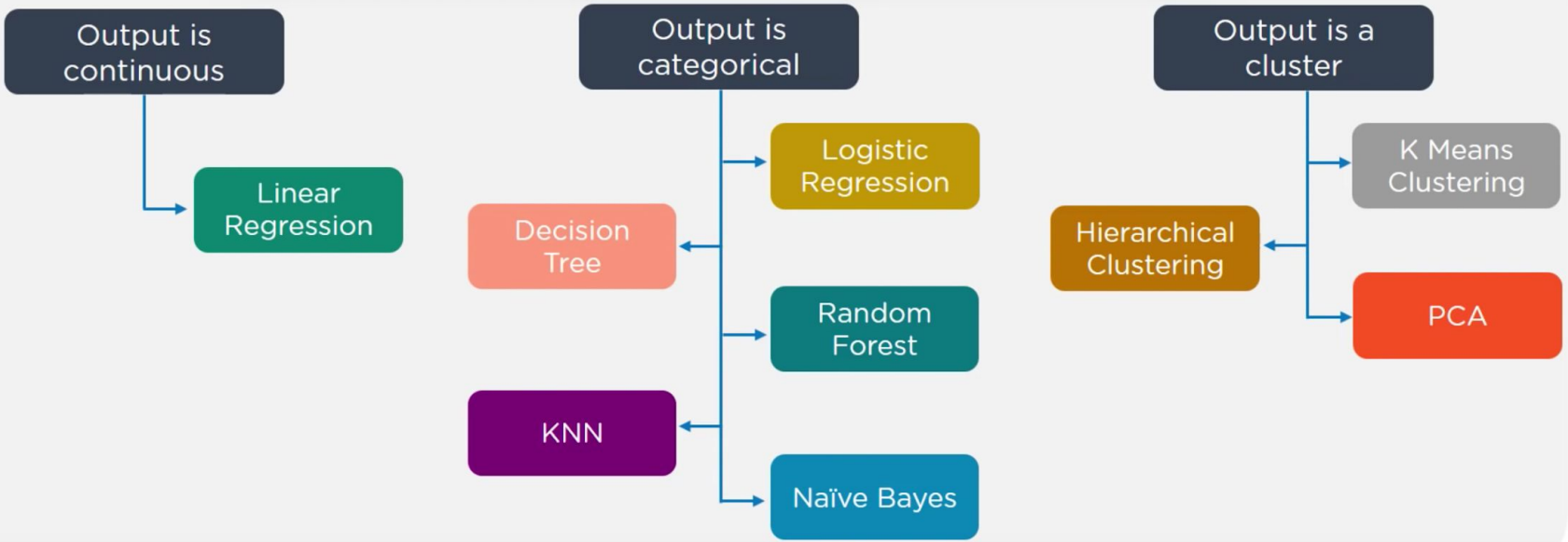
$D(X, ii) = \sqrt{(3-7)^2 + (7-4)^2} = \sqrt{16+9} = 5$

$D(X, iii) = \sqrt{(3-3)^2 + (7-4)^2} = 3 \longrightarrow N1 \rightarrow TRUE$

$D(X, iv) = \sqrt{(3-1)^2 + (7-4)^2} = \sqrt{4+9} = 3.6 \rightarrow N2 \rightarrow TRUE$

2 TRUE > 1 FALSE

Considering the long list of Machine Learning algorithm, given a data set, how do you decide which one to use?



K-NN Regression

Need for K-NN Regression.

Example: What could be the predicted price of buying a house in a locality, if the prices of the neighbouring houses and the parameters that define the prices are known.

Overview

- K Nearest Neighbors is a simple algorithm that stores all available cases and predict the numerical target based on a similarity measure (e.g., distance functions)
- KNN has been used in statistical estimation and pattern recognition already in the beginning of 1970's as a non-parametric technique

- A simple implementation of KNN regression is to **calculate the average of the numerical target of the K nearest neighbors**
- Another approach uses an inverse distance weighted average of the K nearest neighbors
- KNN regression uses the same distance functions as KNN classification

Theory

- X0 – prediction point
- K- k training observations closest to X0
- **Optimal value of K?**
- $$\hat{y} = f(x) = \frac{1}{k} \sum_{j=1}^{k} y_{i_j}$$

Most Common Function: **Average**

Q #1

- K Nearest Neighbours Regression

Age	Weight
20	50
24	54
25	52
	60

Smoothing function is **'Average'**

K = 1, Age = 24, nearest neighbour of 56

K = 2, Age = 24.5

K = 3, Age = 26.3

- What is the age when Weight is 56?

Q #2

Age	Loan	House Price Index	Distance
25	$40,000	135	102000
35	$60,000	256	82000
45	$80,000	231	62000
20	$20,000	267	122000
35	$120,000	139	22000
52	$18,000	150	124000
23	$95,000	127	47000
40	$62,000	216	80000
60	$100,000	139	42000
48	$220,000	250	78000
33	$150,000	264	8000
48	**$142,000**	**?**	

Here, **Loan** is the parameter for finding the nearest neighbour, not **Age**.

What is the HPI for $142,000 loan?

D = Sqrt[(48-33)^2 + (142000-150000)^2] = 8000.01

Age	Loan	House Price Index	Distance	
25	$40,000	135	102000	
35	$60,000	256	82000	
45	$80,000	231	62000	
20	$20,000	267	122000	
35	$120,000	139	22000	2
52	$18,000	150	124000	
23	$95,000	127	47000	
40	$62,000	216	80000	
60	$100,000	139	42000	3
48	$220,000	250	78000	
33	$150,000	264	8000	1
48	$142,000	?		

Q #2

$$D = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

Euclidean Distance

Age	Loan	House Price Index	Distance	
25	$40,000	135	102000	
35	$60,000	256	82000	
45	$80,000	231	62000	
20	$20,000	267	122000	
35	$120,000	139	22000	2
52	$18,000	150	124000	
23	$95,000	127	47000	
40	$62,000	216	80000	
60	$100,000	139	42000	3
48	$220,000	250	78000	
33	$150,000	264	8000	1
48	**$142,000**	**?**		

- By having K=3, the prediction for HPI is equal to the average of HPI for the top three neighbors
- HPI = (264+139+139)/3 = 180.7

Q #2 : Solved

Problem with KNN Average

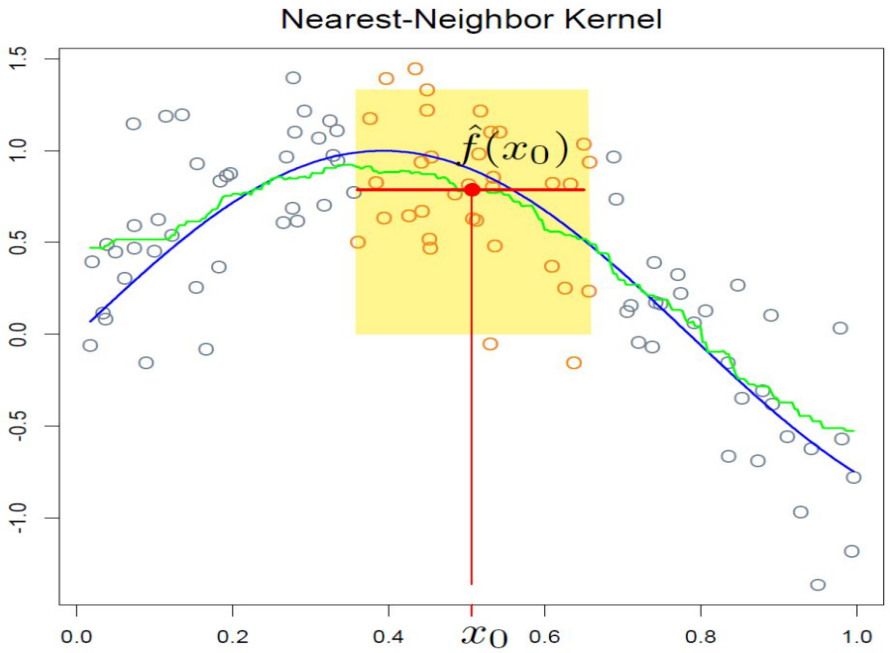
Problem

Regression function $\hat{f}(x)$ is discontinuous - "bumpy".
Neighborhood set changes discontinuously.

Solution

Weigh all points such that their contribution drop off smoothly with distance.

KNN Average Example



True function
KNN average
Observations contributing to $\hat{f}(x_0)$

Kernel Smoothing

Solution to K-NN Average Problem

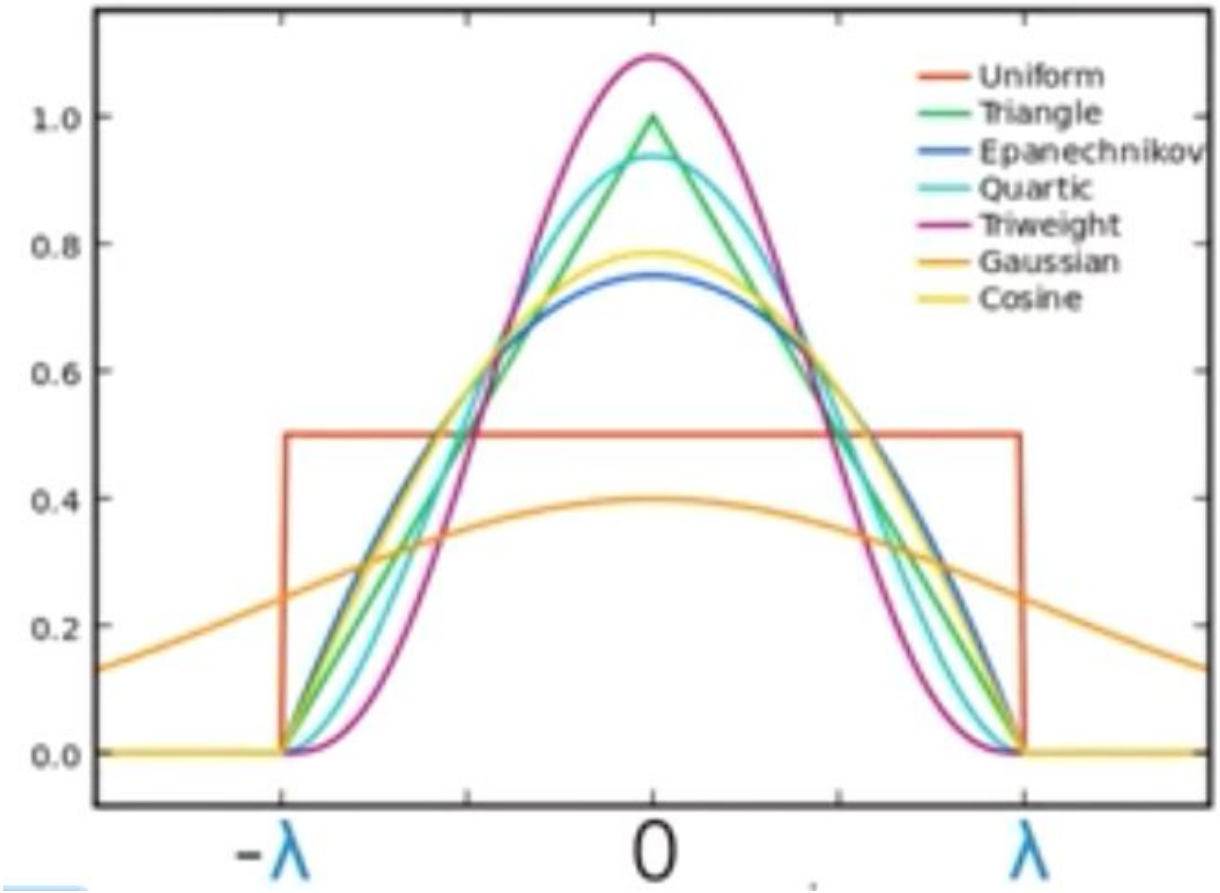
By definition, the kernel is the weighting function.
The goal is to give more importance to closer observations without ignoring observations that are further away.

In Brief

For any query point x_0, the value of the function at that point $f(x_0)$ is some combination of the (nearby) observations, s.t., $f(x)$ is smooth.

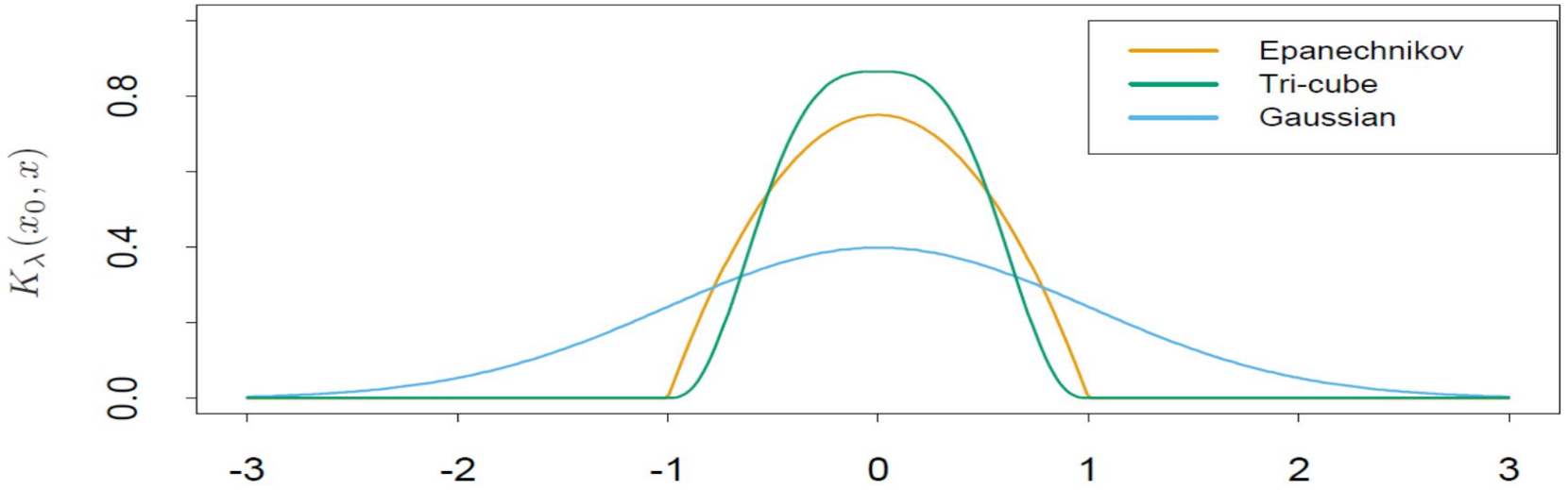
The contribution of each observation $x_i, f(x_i)$ to $f(x_0)$ is calculated using a weighting function *or* Kernel $K_\lambda(x_0, x_i)$.

λ - the width of the neighborhood



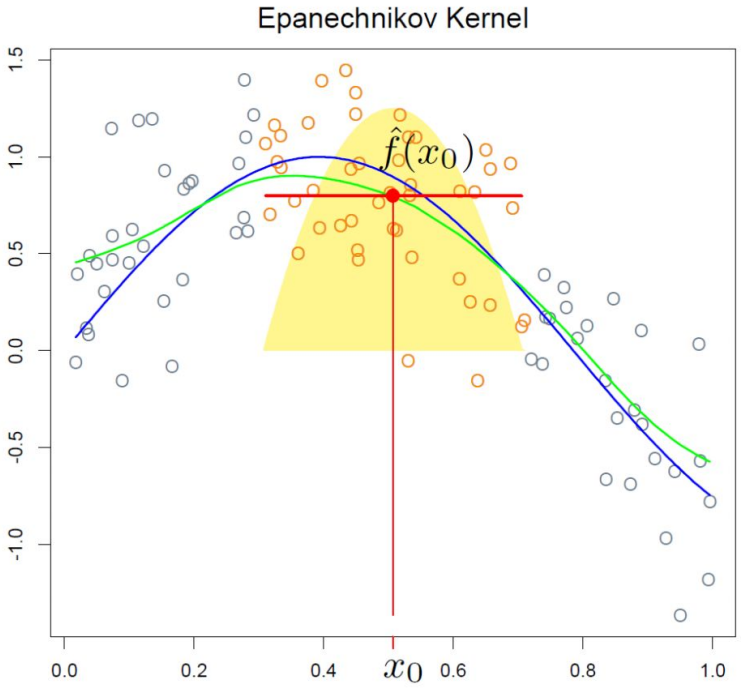
λ - the width of the neighborhood

Popular Kernels



Epanechnikov	*Compact* (only local observations have non-zero weight)
Tri-cube	Compact and differentiable at boundary
Gaussian density	Non-compact (all observations have non-zero weight)

Epanechnikov Quadratic Kernel Example



Estimated function is smooth
Yellow area indicates the weight assigned to observations in that region.

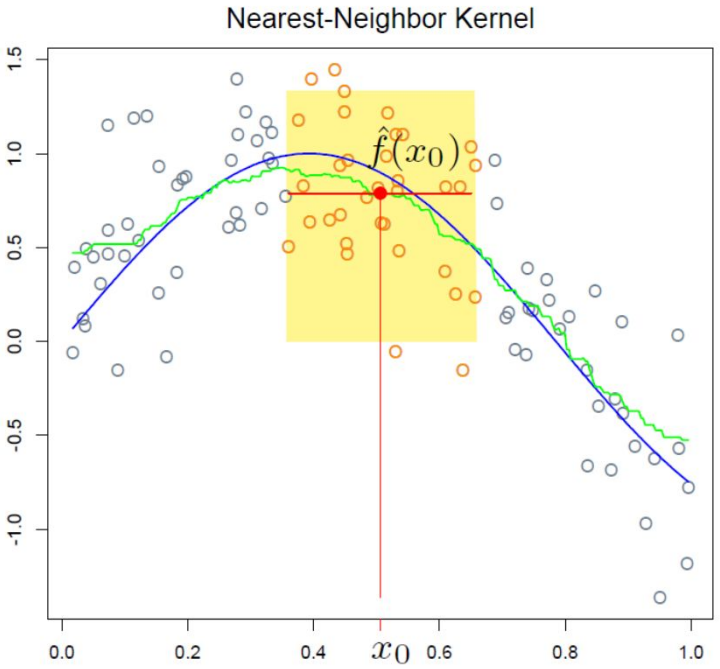
Epanechnikov Quadratic Kernel Equations

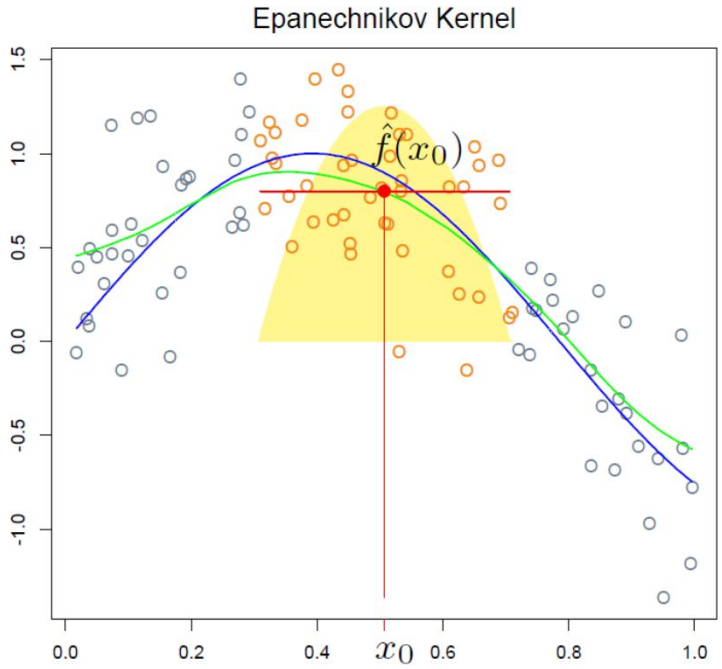
$$\hat{f}(x_0) = \frac{\sum_{i=1}^{N} K_\lambda(x_0, x_i) y_i}{\sum_{i=1}^{N} K_\lambda(x_0, x_i)}$$

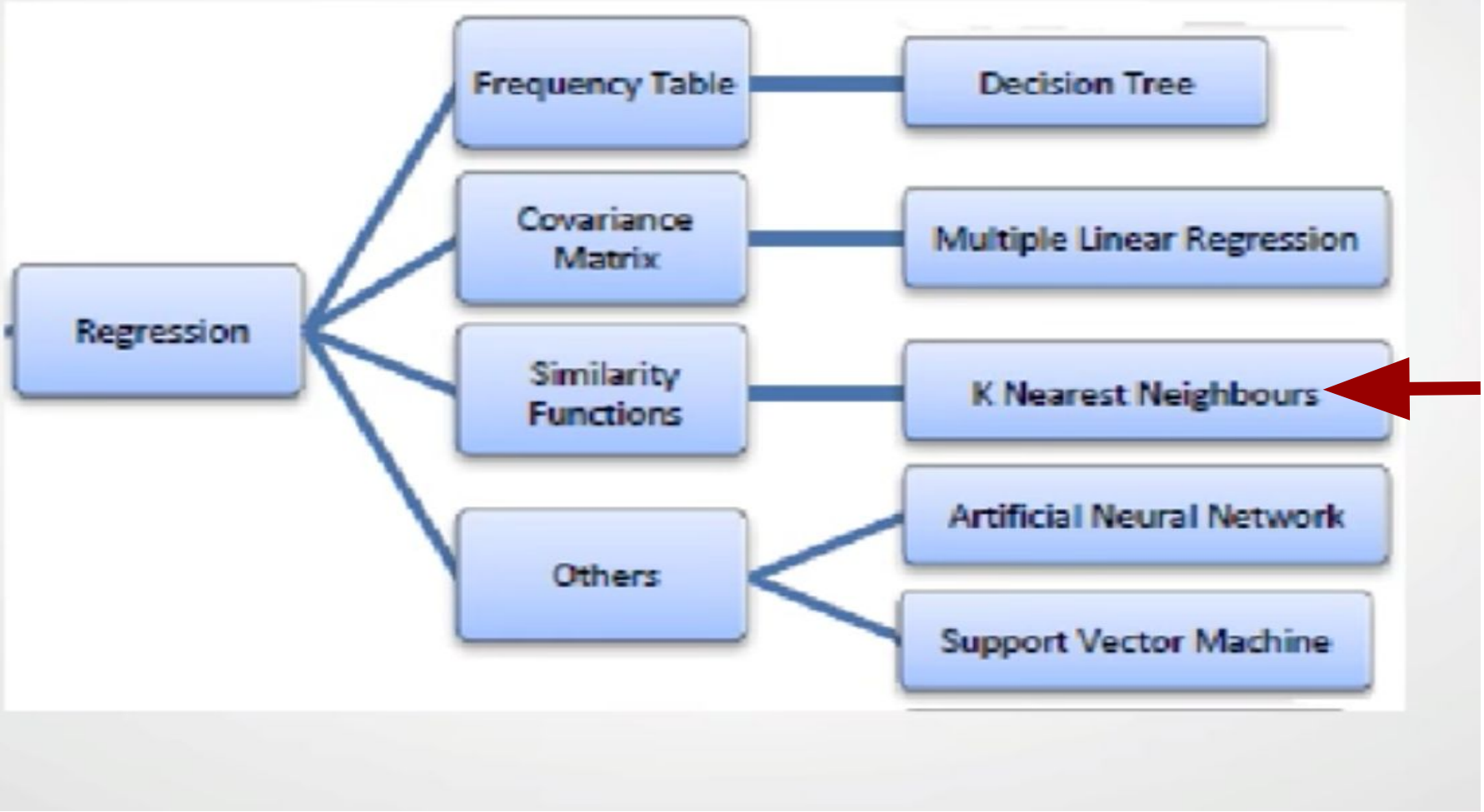
$$K_\lambda(x_0, x) = D\left(\frac{|x - x_0|}{\lambda}\right)$$

$$D(t) = \begin{cases} \frac{3}{4}(1 - t^2) & \text{if} |t| \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

KNN vs Smooth Kernel Comparison






Regression
Frequency Table
Covariance Matrix
Similarity Functions
Others
Decision Tree
Multiple Linear Regression
K Nearest Neighbours
Artificial Neural Network
Support Vector Machine

How is it related to Linear Regression

- KNN has higher prediction power as compare to Linear regression – takes care of the non-linearity

  Reason: Non-parametric method. In Linear Regression the model fits into a mathematical linear equation. In K-NN there is no pre-trained model

- However, if the linear regression function is close to reality then it will perform better

K-D Trees

(K Dimensional Trees: Here k =2 implying Binary

K-D Trees)

Need for K-D Trees

- Computing the distances between all pairs of points is very computationally expensive. Its **O(dN)**, '**d**' is the dimension/number of features & '**N**' number of samples
- Designing an efficient data structure can reduce the computational overhead a lot.
- For the problem of finding nearest neighbours the data structure of choice is the **K-D Tree**.
- It has been around since the late 1970s, when it was devised by Friedman and Bentley.
- It reduces the cost of finding a nearest neighbour to **O(d*logN)**.
- **Note**: Most of the computation is required for calculating the mean, which in turn requires a sort.

Q #1:

For the Data-points:

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

Find the nearest neighbour for

a) 4.5,2

b) 3,5

Use K-D Trees Algorithm.

Q #1: $(5,4),(1,6),(6,1),(7,5),(3,7),(2,2),(6,8)$

Calculate the **mean**.

It is the datapoint **(4.28, 4.8).**

Choose the point closest to the mean from the given dataset. Here it is **(5,4)**

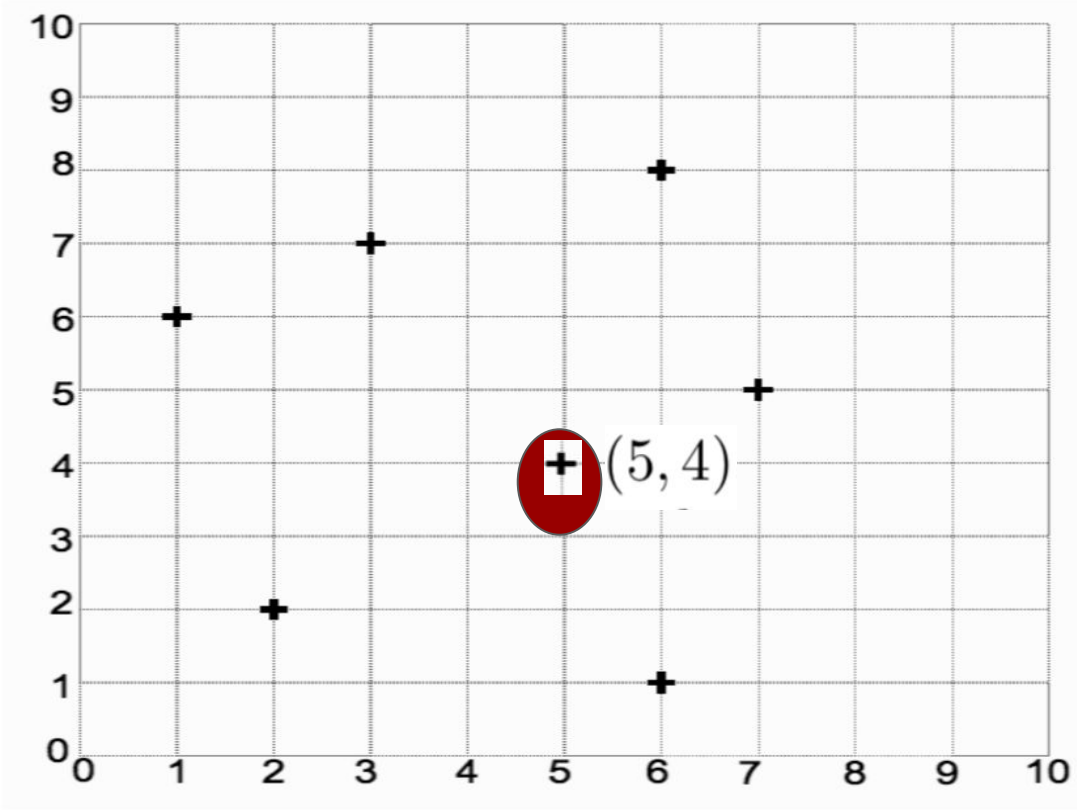


FIGURE 7.5 The initial set of 2D data.

Question from Stephen Marsland Textbook.

Note: Correction of data point as **(3,7) and (6,8)** instead of **(2,7) and (5,8)** as given in the textbook.

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

Q #1:

4 , 1 , 5 , 7 , 3 , 2 , 6

Order the data points - first based on its ‘x’ values. If ‘x’ value is same order based on ‘y’ values.

K-D Tree is usually a ‘binary tree’. Therefore, choose the mean datapoint as **‘root’**, the lowest value as ‘**left node**’ and largest value as ‘**right node**’.

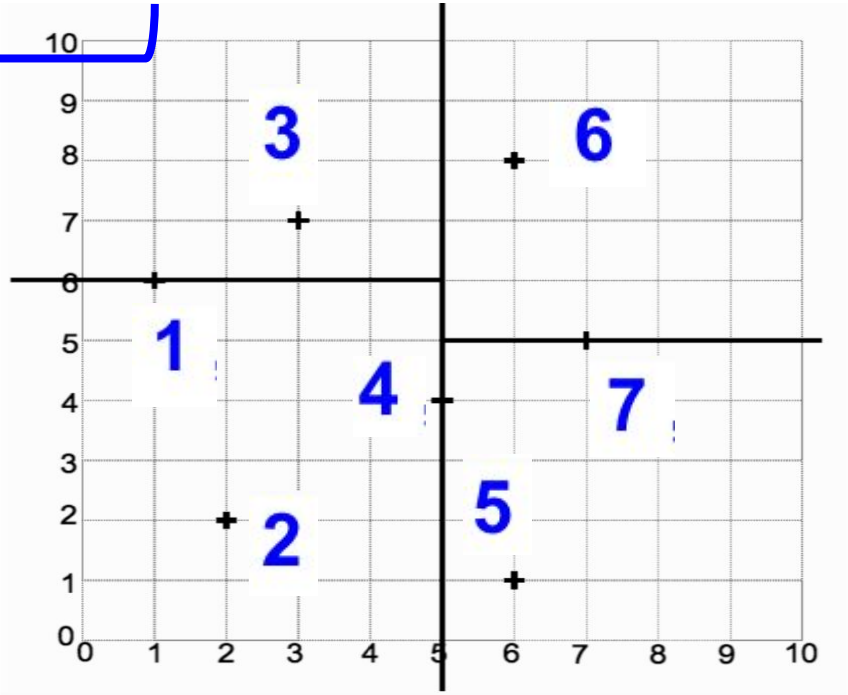
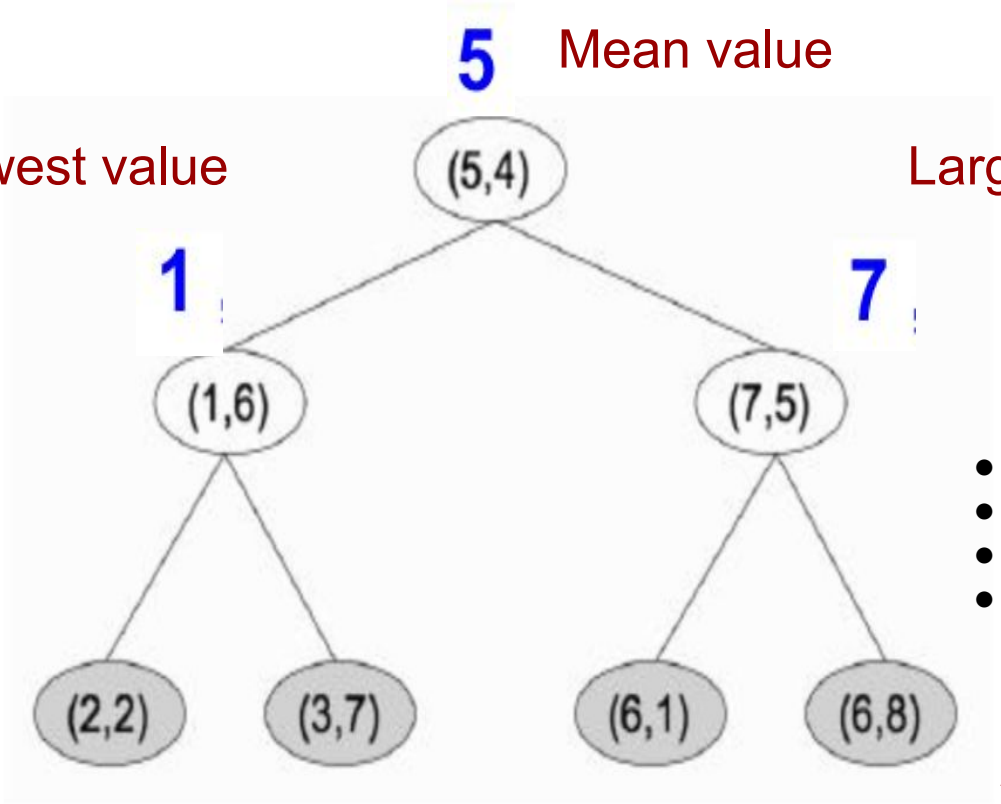


FIGURE 7.6 The splits and leaf points found by the KD-tree.

Q #1:



- **Point (2,2)**
- X value 2<5 : go left
- Ignore x value.
- Y value 2<6 : insert to left of 1,6

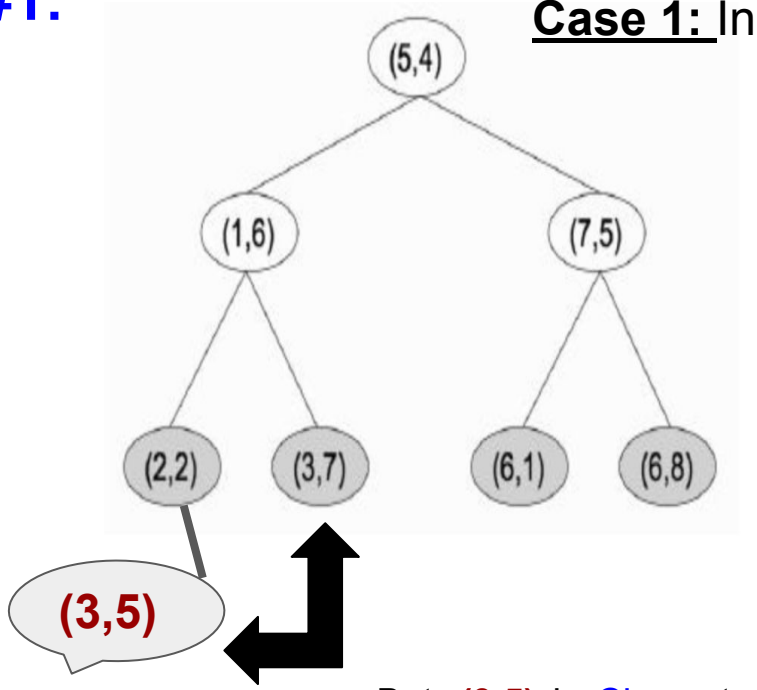
- **Point (6,8)**
- X value 6>5 : go right
- Ignore x value.
- Y value 8>5 : insert to right of 7,5

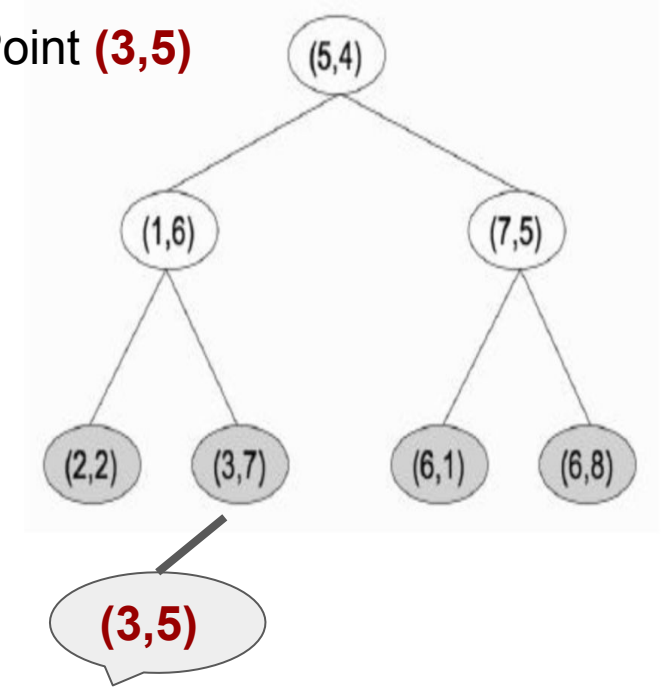
At this level, Leaf Nodes are created based on the '**y' value.**

FIGURE 7.7 The KD-tree that made the splits.

Q #1:

Case 1: Insert Test Point **(3,5)**





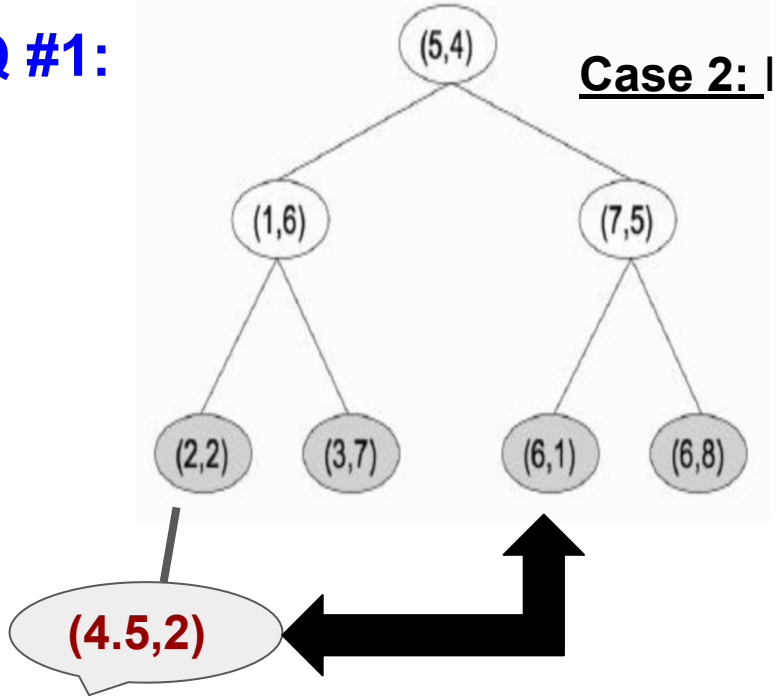
- **Point (3,5)**
- X value 3<5 : go left
- Ignore x value.
- Y value 5>2 : insert to right of 2,2

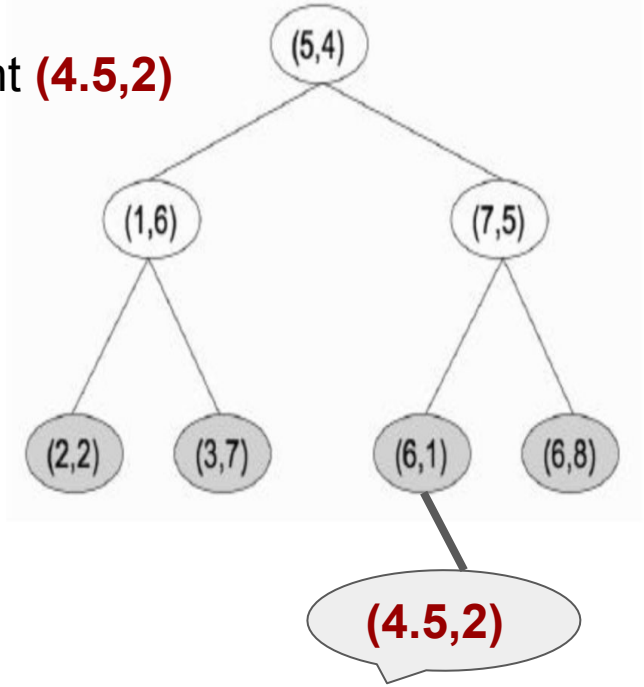
But, **(3,5)** is Closer to **(3,7)** compared to (2,2)

Therefore, in K-D trees, since there is a possibility that if only the parent node is checked while inserting the new test point, the sibling could be too far away, the K-D Tree Algorithm ***needs to check the sibling nodes also and not just the parent node.***

Q #1:

Case 2: Insert Test Point **(4.5,2)**





- **Point (4.5,2)**
- X value 4.5<5 : go left
- Ignore x value.
- Y value 2<6 : insert to left of 2,2

But, **(4.5,2)** is Closer to **(6,1)** compared to (2,2)

Therefore, in K-D trees, since there is a possibility that if only the parent node is checked while inserting the new test point, the sibling could be too far away, the K-D Tree Algorithm ***needs to check the sibling nodes also and not just the parent node.***

Q #1 Solved

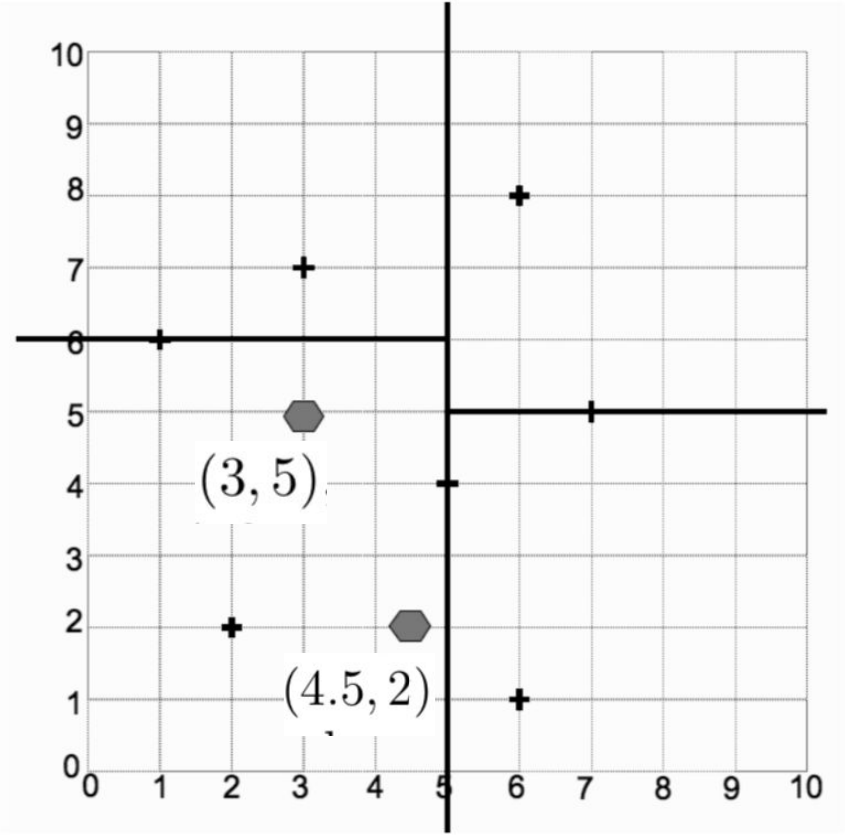


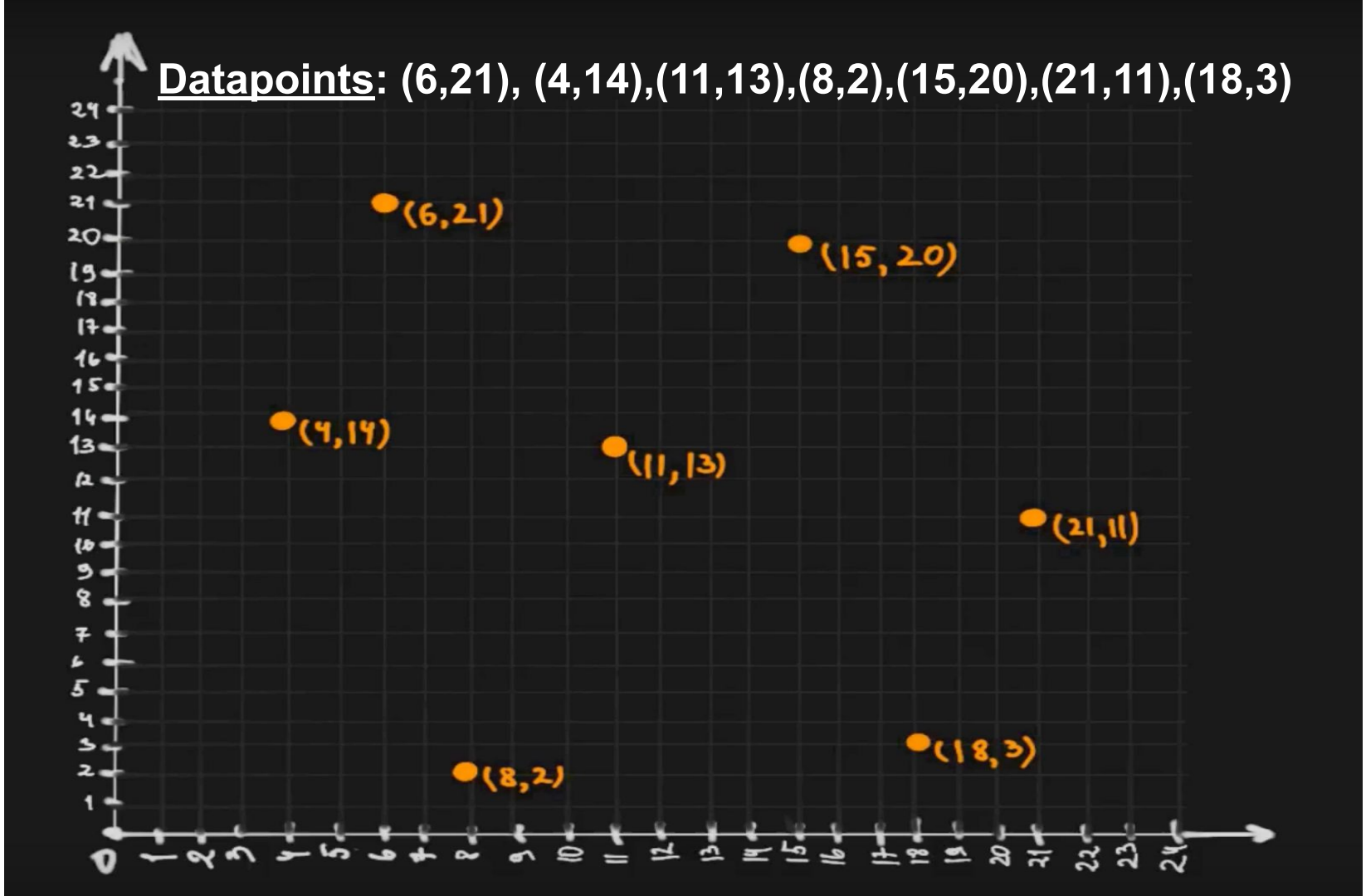
FIGURE 7.8 Two test points for the example KD-tree.

Q #2:

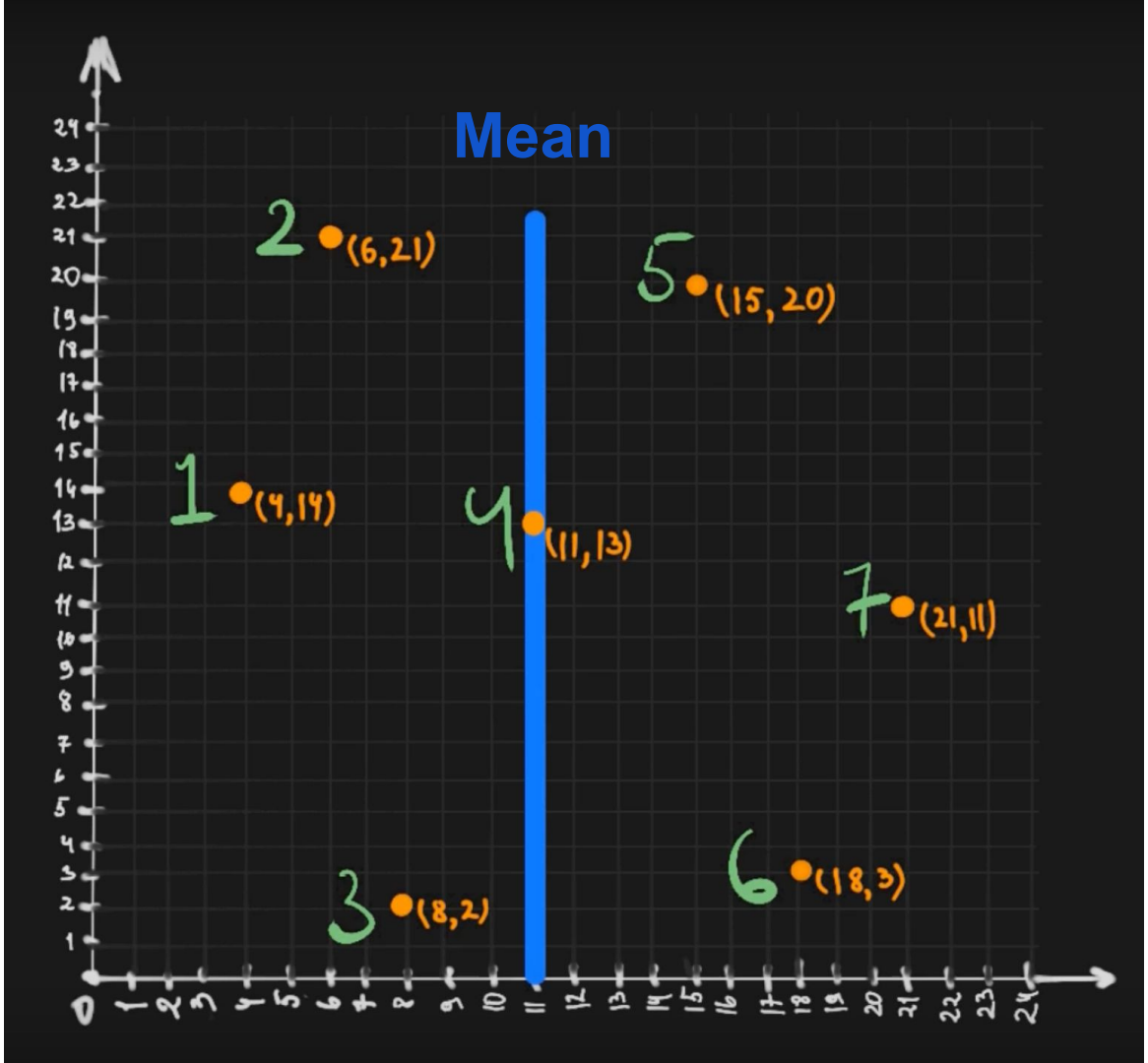
For the Data-points:
(6,21),(4,14),(11,13),(8,2),(15,20),(21,11),(18,3),

Find the nearest neighbour for (9,19). Use K-D Trees Algorithm.

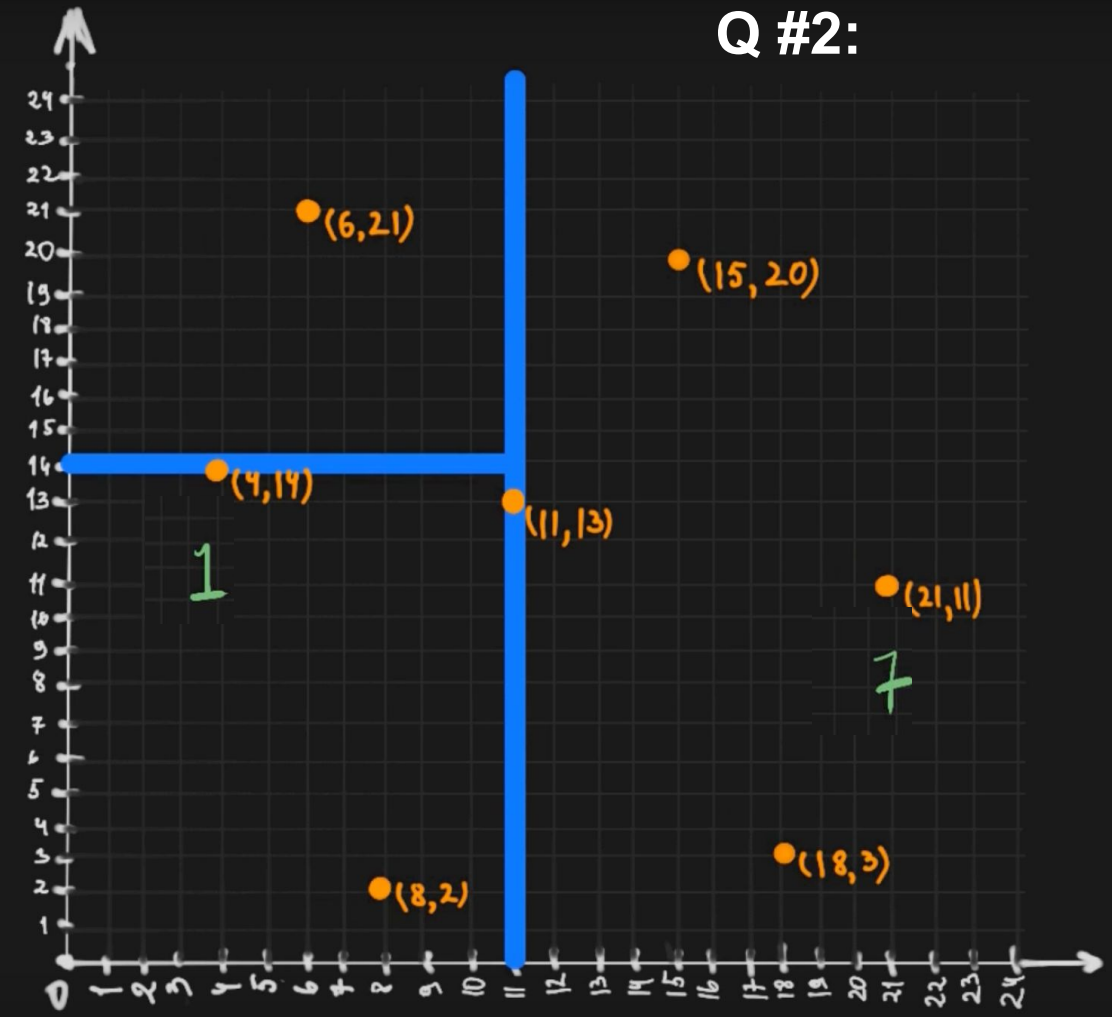
Q #2: Find the nearest neighbour for 9,19

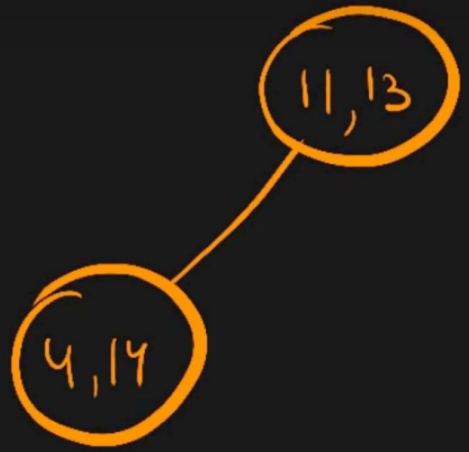


Q #2:

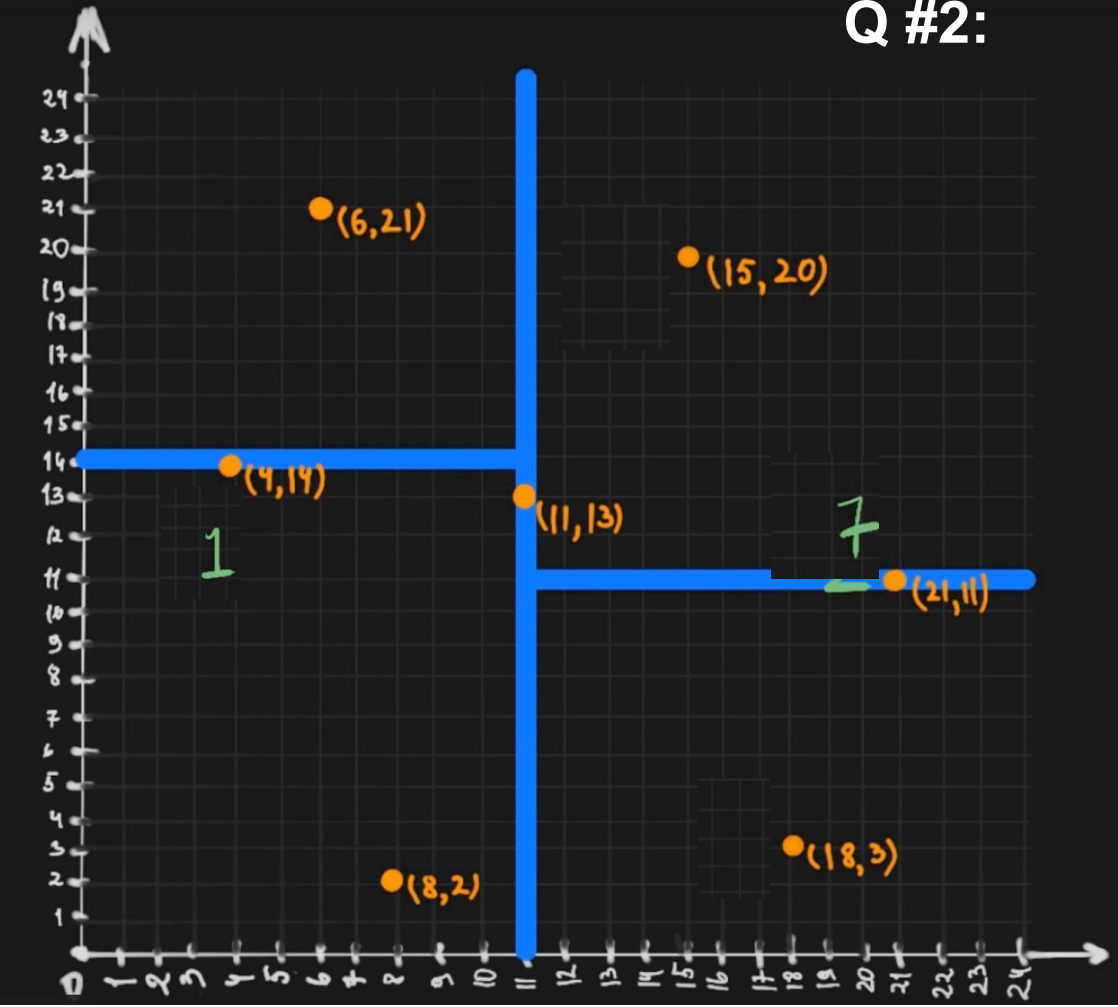


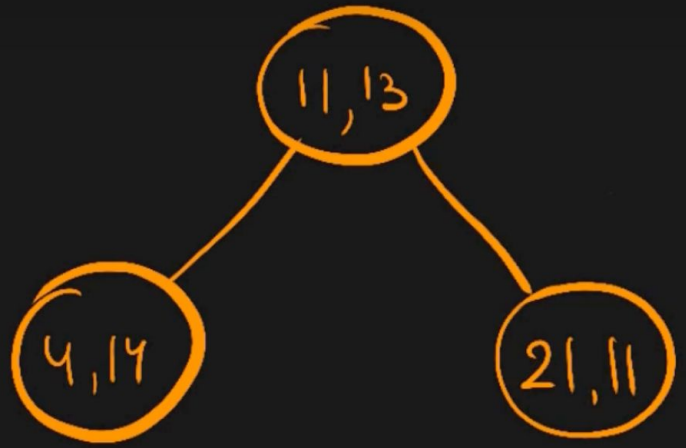
Q #2:



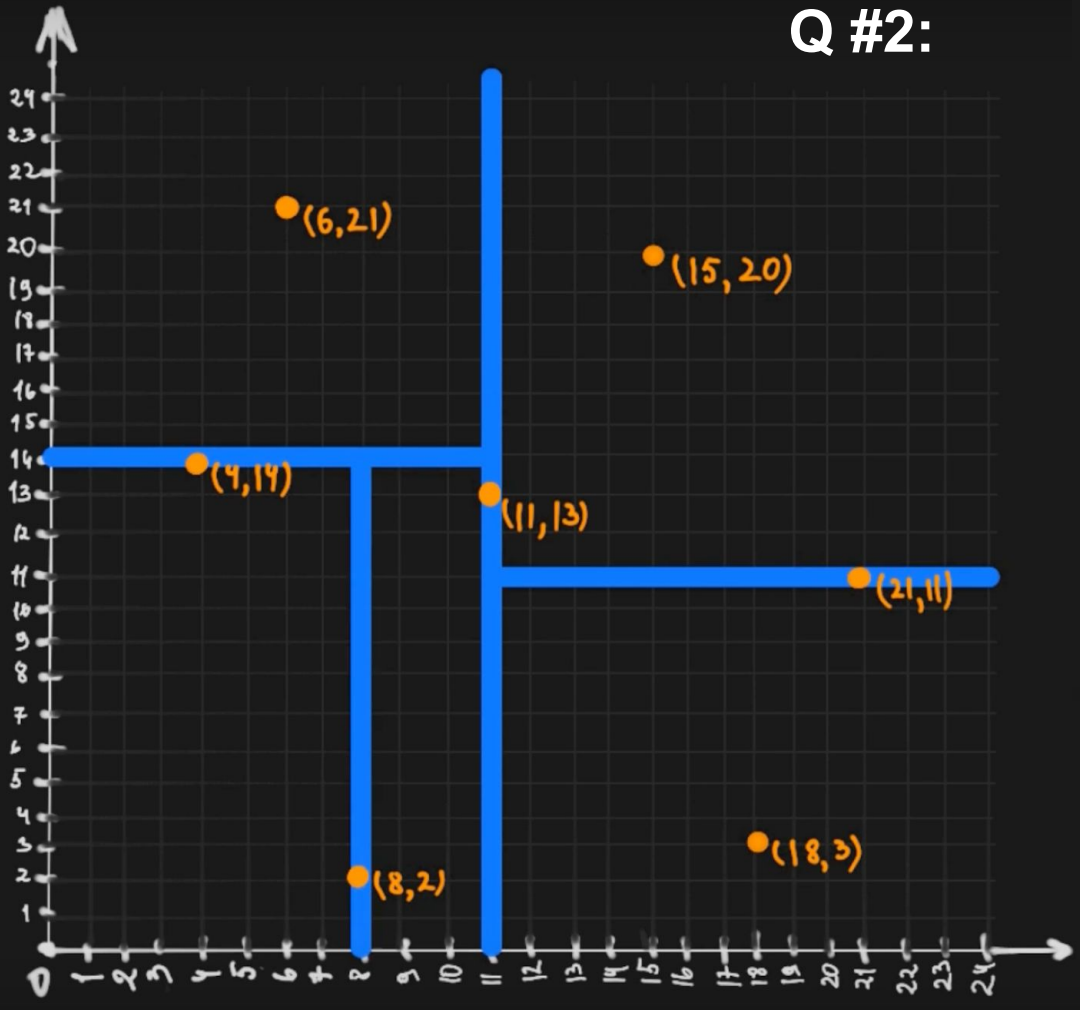


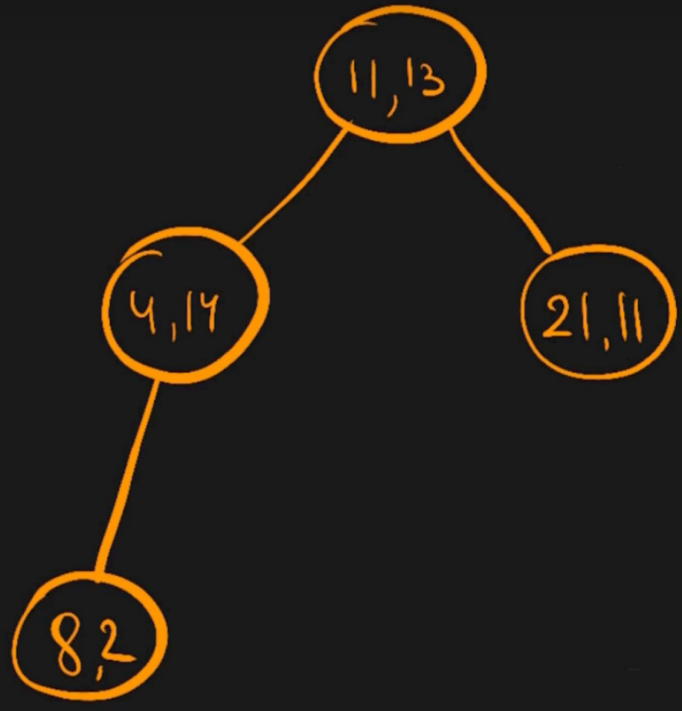
Q #2:



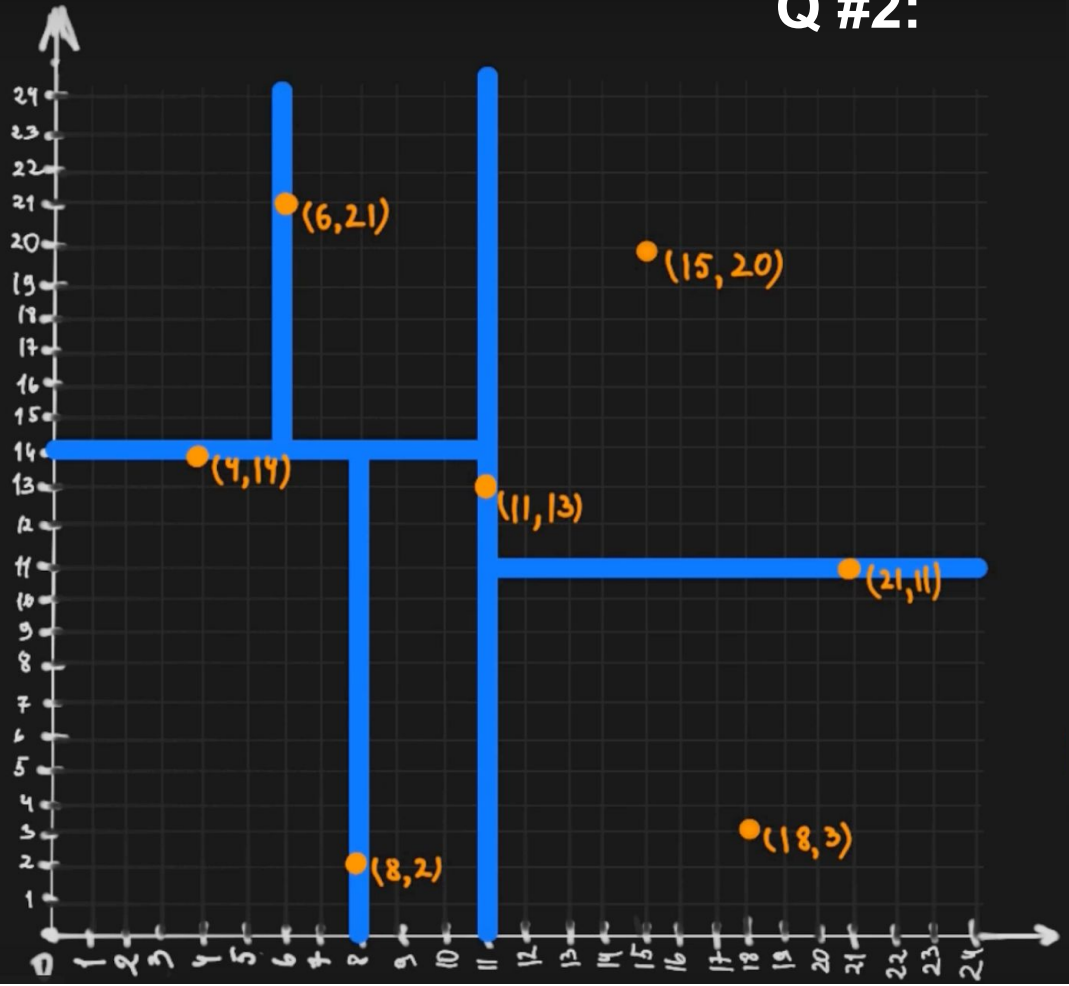


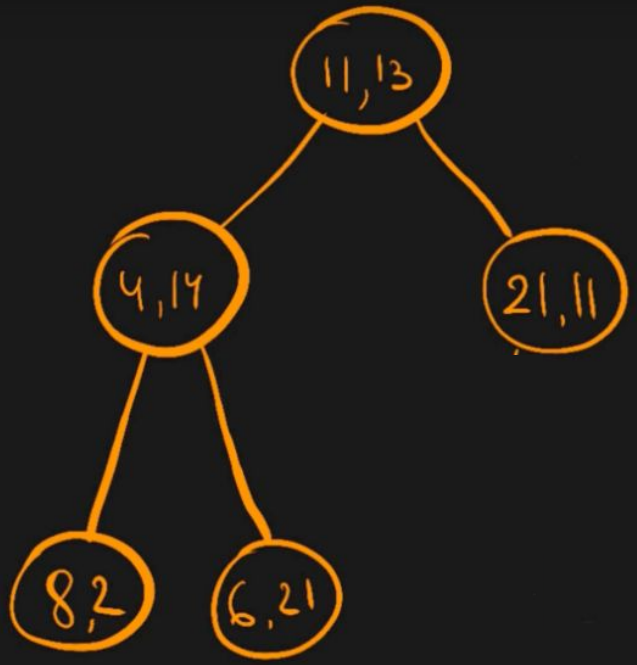
Q #2:



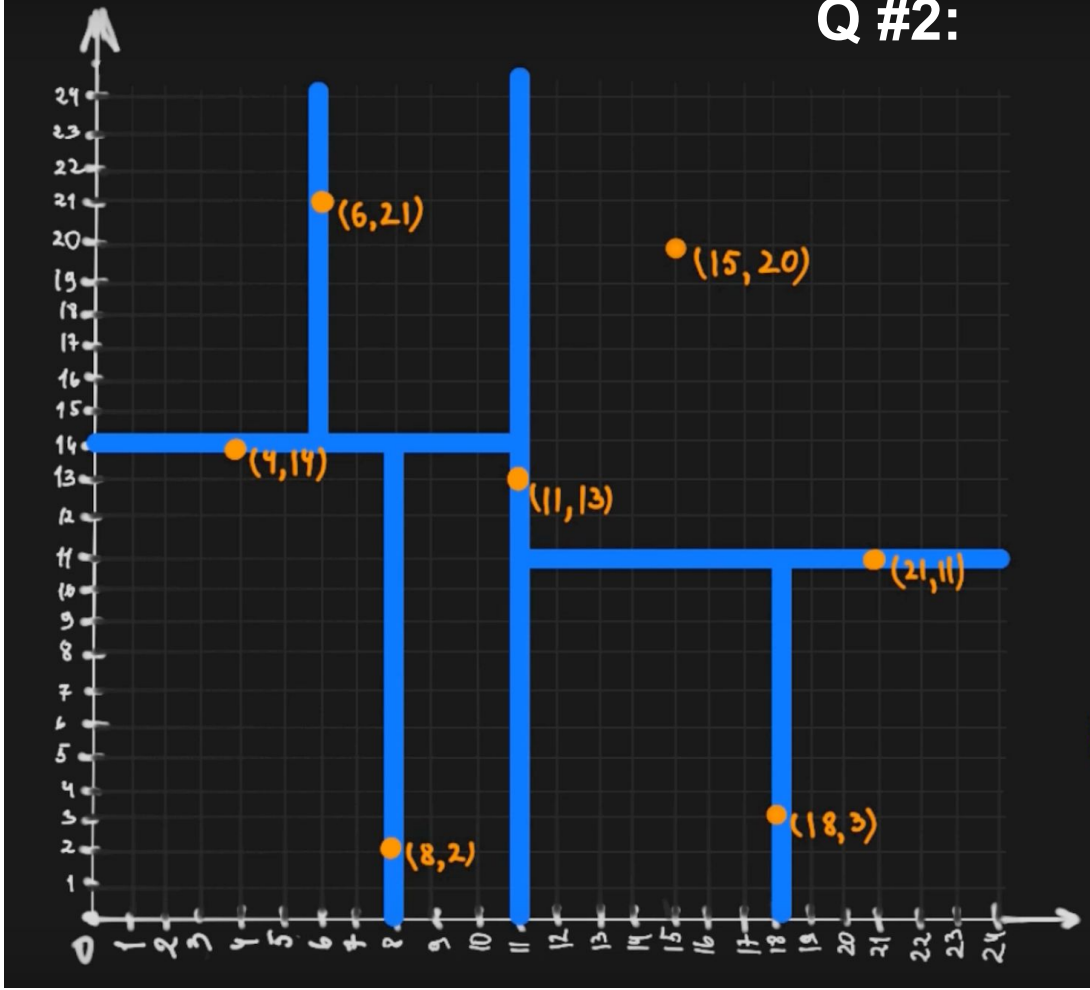


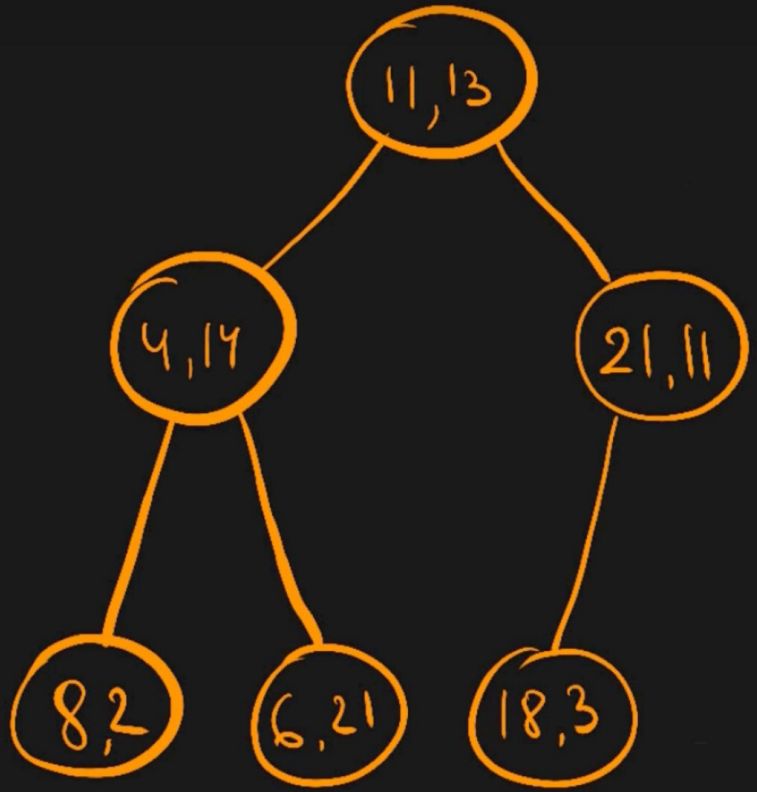
Q #2:



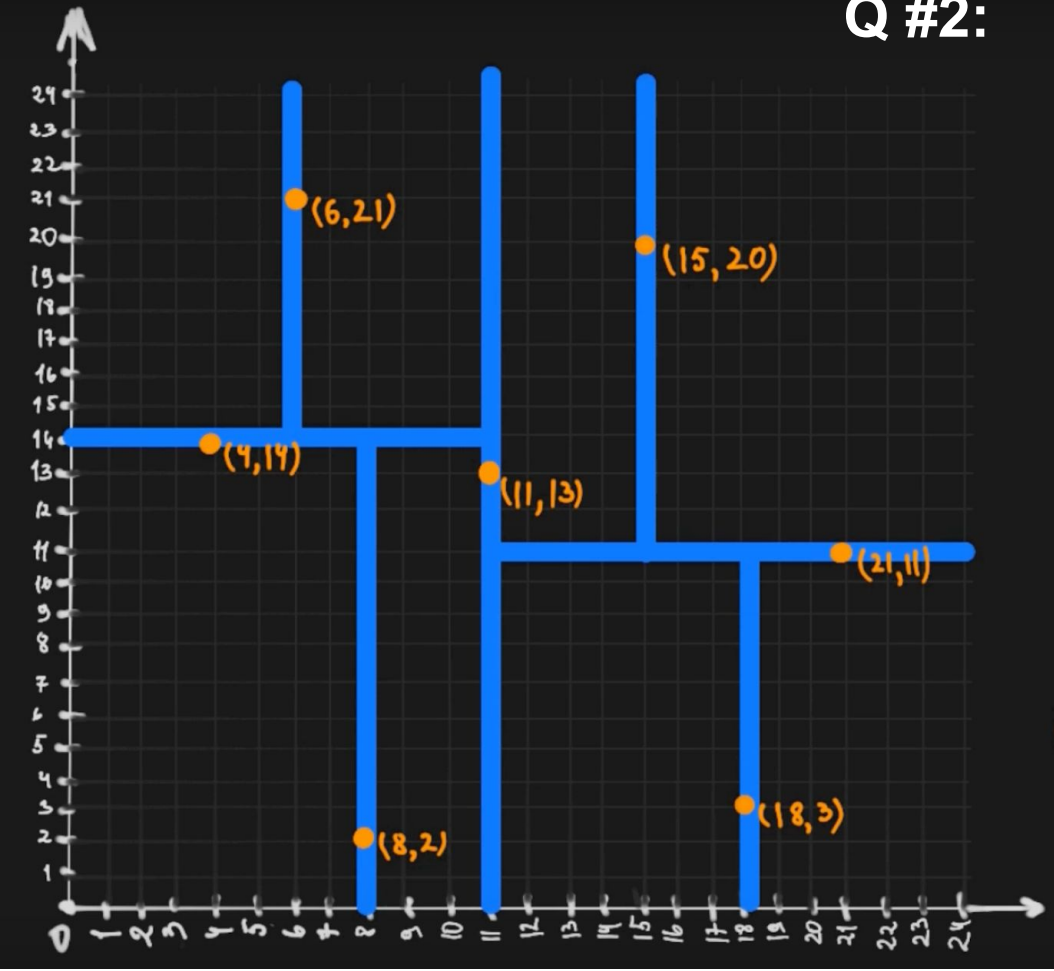


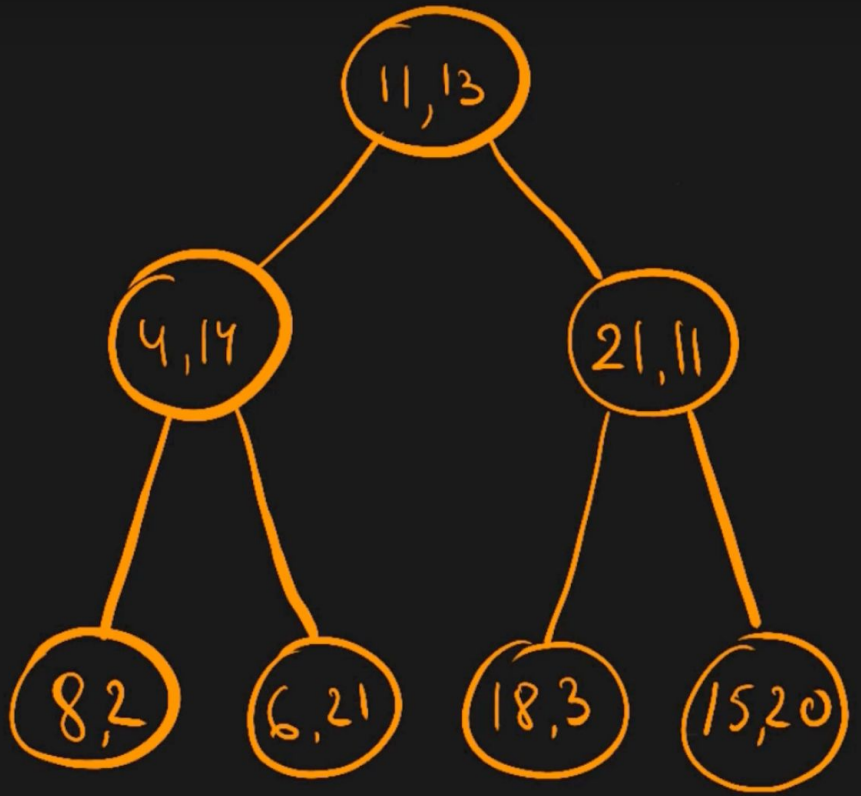
Q #2:





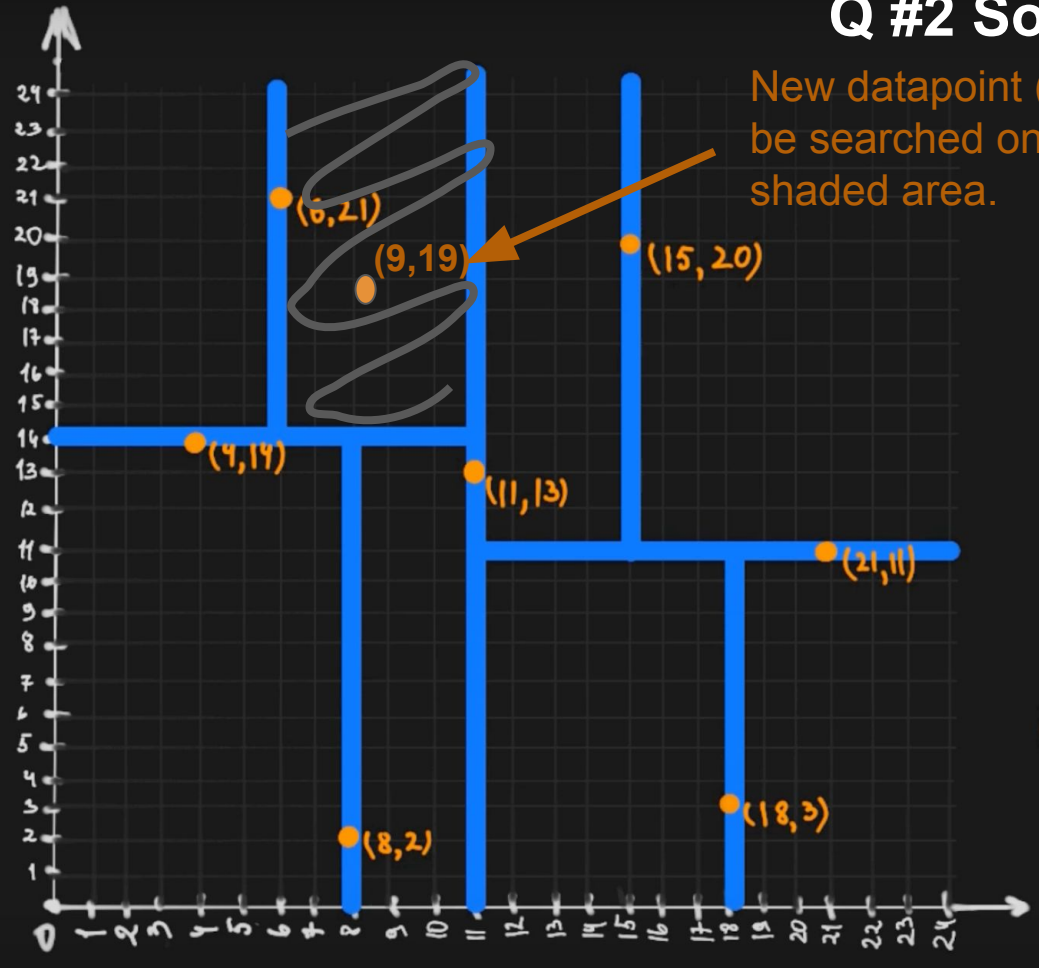
Q #2:

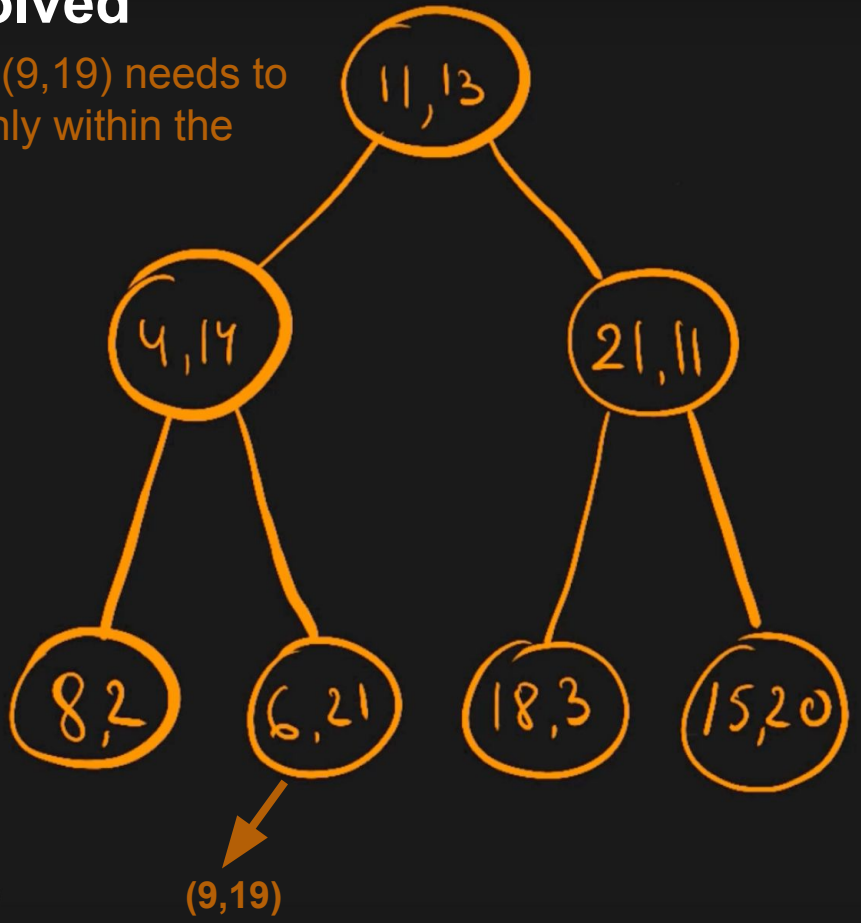




Q #2 Solved

New datapoint (9,19) needs to be searched only within the shaded area.





Explanation of K-D Tree Algorithm

one dimension is split in each step, and the position of the split is found by computing the median of the points that are to be split in that one dimension, and putting the line there. In general, the choice of which dimension to split alternates through the different choices, or it can be made randomly. The algorithm below cycles through the possible dimensions based on the depth of the tree so far, so that in two dimensions it alternates horizontal and vertical splits.

The centre of the construction method is simply a recursive function that picks the axis to split on, finds the median value on that axis, and separates the points according to that value

In K-D trees, since there is a possibility that if only the parent node is checked while inserting the new test point, the sibling could be too far away, the K-D Tree Algorithm ***needs to check the sibling nodes also and not just the parent node.***

kNN pros and cons

- Almost no assumptions about the data
  - smoothness: nearby regions of space → same class
  - assumptions implied by distance function (only locally!)
  - non-parametric approach: "let the data speak for itself"
    - nothing to infer from the data, except *k* and possibly D()
    - easy to update in online setting: just add new item to training set
- Need to handle missing data: fill-in or create a special distance
- Sensitive to class-outliers (mislabeled training instances)
- Sensitive to lots of irrelevant attributes (affect distance)
- Computationally expensive:
  - space: need to store all training examples
  - time: need to compute distance to all examples: O(*nd*)
    - *n* … number of training examples, *d* … cost of computing distance
    - *n* grows → system will become slower and slower
    - expense is at *testing*, not *training* time (bad)

Summary: kNN

- Key idea: nearby points → same class
  - important to select good distance function
- Can be used for classification and regression
- Simple, non-linear, asymptotically optimal
  - does not make assumptions about the data
  - "let the data speak for itself"
- Select k by optimizing error on held-out set
- Naïve implementations slow for big datasets
  - use K-D trees (low-d) or inverted lists (high-d)

Making kNN fast

- Training: O(1), but testing: O(*nd*)
- Reduce *d*: dimensionality reduction
  - simple feature selection, other methods O(d^3)
- Reduce *n*: don't compare to **all** training examples
  - idea: quickly identify $m \ll n$ potential near neighbors
    - compare only to those, pick *k* nearest neighbors → O(*md*) time
  - **K-D trees**: low-dimensional, real-valued data
    - O (*d* log *n*), only works when $d \ll n$, inexact: may miss neighbors
  - **inverted lists**: high-dimensional, discrete data
    - O ($d'n'$) where $d' \ll d$, $n' \ll n$, only for sparse data (e.g. text), exact
  - **fingerprinting**: high-d, sparse or dense
    - O(dn'), $n' \ll n$ … bits in fingerprint, inexact: may miss near neighbors