# K-Nearest Neighbors (KNN)

- K-Nearest Neighbors (KNN) is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining and intrusion detection

- It is widely disposable in real-life scenarios since it is non-parametric, meaning, it does not make any underlying assumptions about the distribution of data

- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories

# KNN

- It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset

- Predictions are made for a new instance (x) by searching through the entire training set for the K most similar instances (the neighbors) and summarizing the output variable for those K instances

- To determine which of the K instances in the training dataset are most similar to a new input a distance measure is used. For real-valued input variables, the most popular distance measure is Euclidean distance

# KNN

- Euclidean distance: It is calculated as the square root of the sum of the squared differences between a new point (x) and an existing point (xi) across all input attributes j

  Euclidean Distance(x, xi) = sqrt( sum( (xj – xij)^2 ) )

Other popular distance measures include:

- Hamming Distance: Calculate the distance between binary vector

- Manhattan Distance: Calculate the distance between real vectors using the sum of their absolute difference. Also called City Block Distance

- Minkowski Distance: Generalization of Euclidean and Manhattan distance

# KNN

KNN can be used for regression and classification problems.

## KNN for Regression

When KNN is used for regression problems the prediction is based on the mean or the median of the K-most similar instances.

## KNN for Classification

When KNN is used for classification, the output can be calculated as the class with the highest frequency from the K-most similar instances. Each instance in essence votes for their class and the class with the most votes is taken as the prediction.

# KNN

## The KNN Algorithm

1. Load the data
2. Initialize K to your chosen number of neighbors
3. For each example in the data
    1. Calculate the distance between the query example and the current example from the data.
    2. Add the distance and the index of the example to an ordered collection
4. Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances
5. Pick the first K entries from the sorted collection
6. Get the labels of the selected K entries
7. If regression, return the mean of the K labels
8. If classification, return the mode of the K labels

# KNN

## How to select the value of K in the K-NN Algorithm?

Below are some points to remember while selecting the value of K in the K-NN algorithm:

- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5
- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model
- Large values for K are good, but it may find some difficulties

# KNN

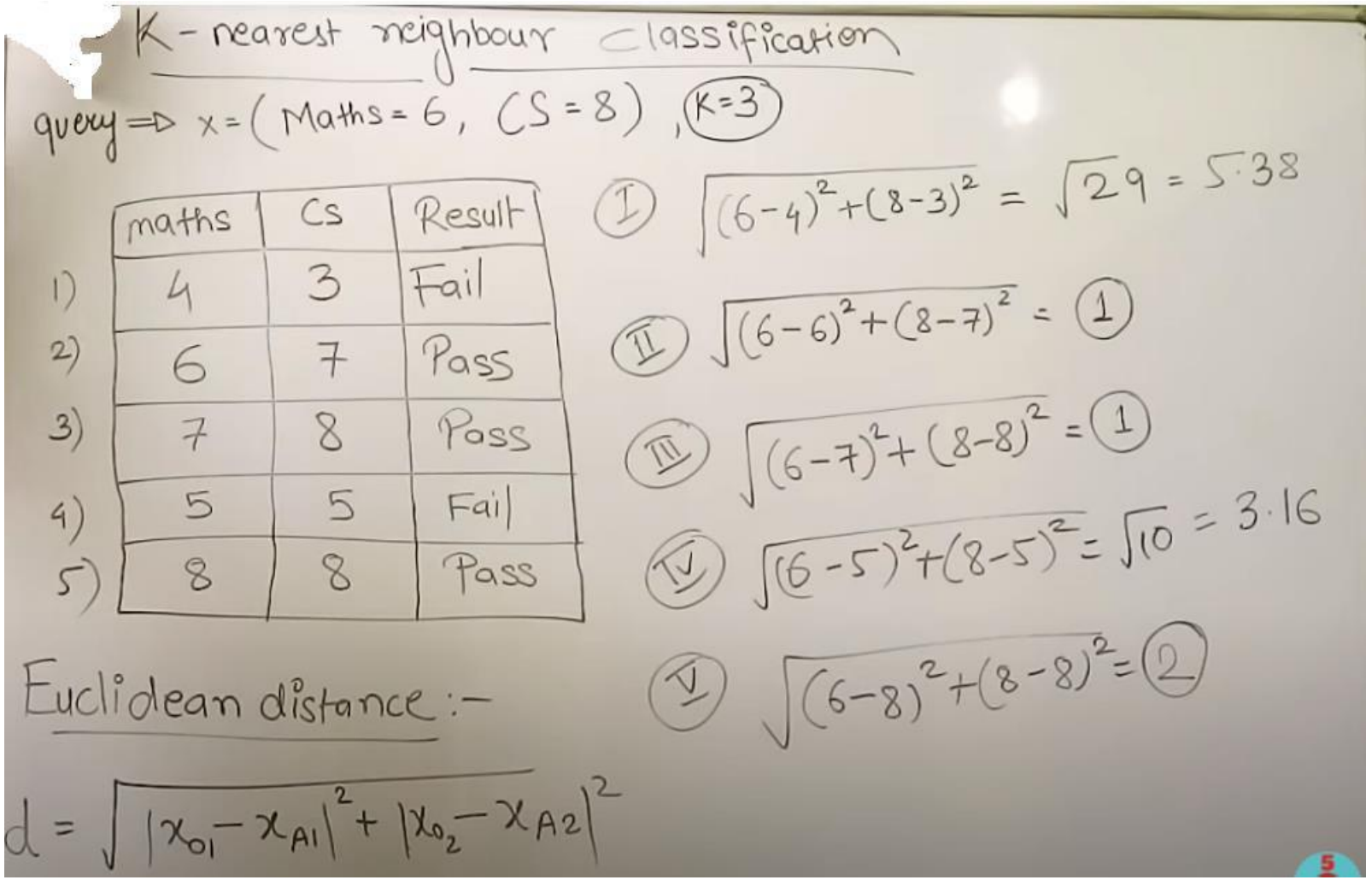
## Advantages

- The algorithm is simple and easy to implement
- There's no need to build a model, tune several parameters, or make additional assumptions
- The algorithm is versatile. It can be used for classification, regression, and search
- It is robust to the noisy training data
- It can be more effective if the training data is large

## Disadvantages

- The algorithm gets significantly slower as the number of examples and/or predictors/independent variables increase
- Always needs to determine the value of K which may be complex some time
- The computation cost is high because of calculating the distance between the data points for all the training samples

# KNN


K-nearest neighbour classification
query ⇒ x = (Maths = 6, CS = 8), (K=3)
maths | Cs | Result
1) 4 | 3 | Fail
2) 6 | 7 | Pass
3) 7 | 8 | Pass
4) 5 | 5 | Fail
5) 8 | 8 | Pass
I $\sqrt{(6-4)^2+(8-3)^2} = \sqrt{29} = 5.38$
II $\sqrt{(6-6)^2+(8-7)^2} = 1$
III $\sqrt{(6-7)^2+(8-8)^2} = 1$
IV $\sqrt{(6-5)^2+(8-5)^2} = \sqrt{10} = 3.16$
V $\sqrt{(6-8)^2+(8-8)^2} = 2$
Euclidean distance :-
$d = \sqrt{|x_{01}-x_{A1}|^2 + |x_{02}-x_{A2}|^2}$

# KNN

- K Nearest Neighbours Regression

| Age | Weight |
|---|---|
| 20 | 50 |
| 24 | 54 |
| 25 | 52 |
| 30 | 60 |

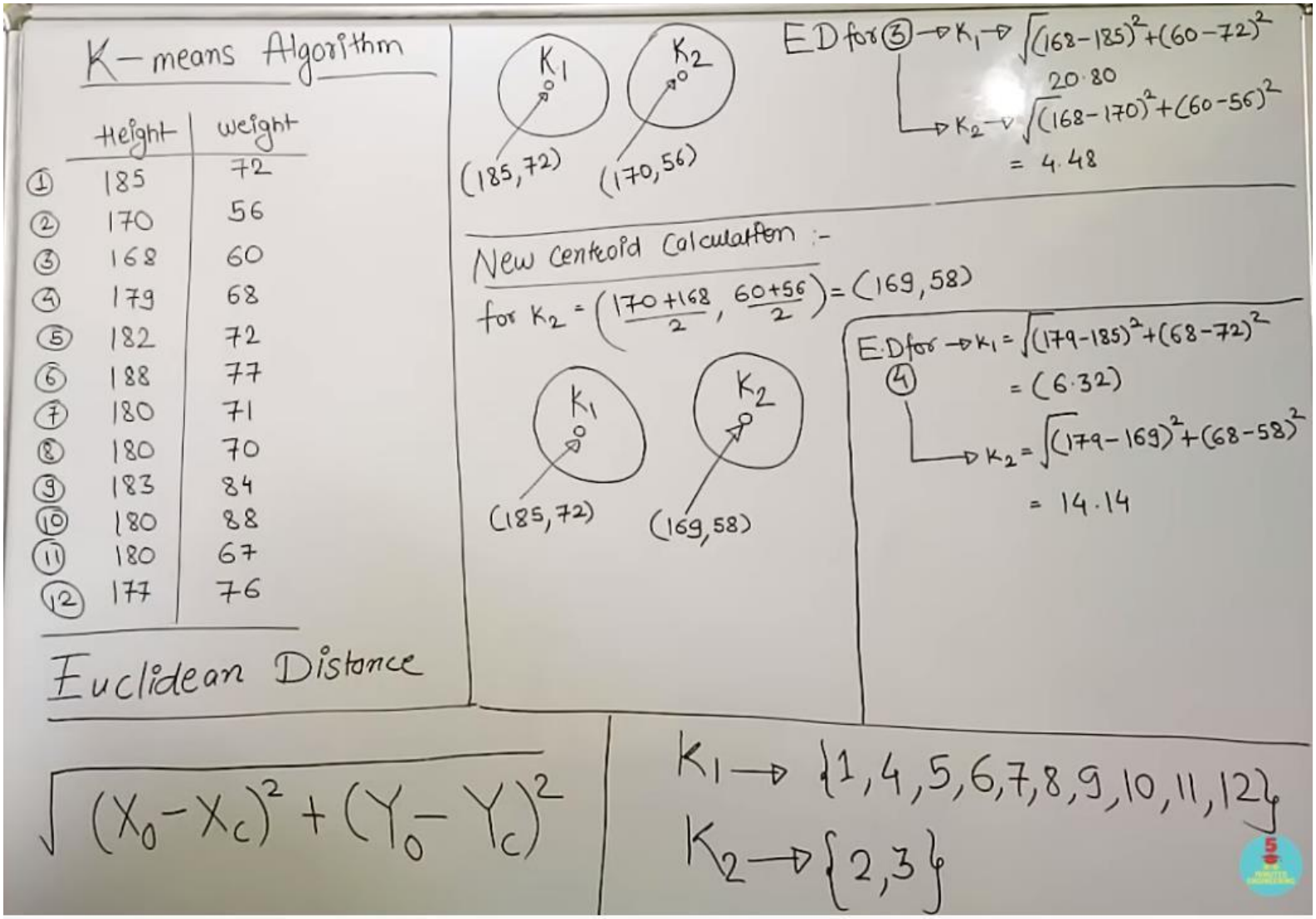
- What is the age when Weight is 56?

# KNN

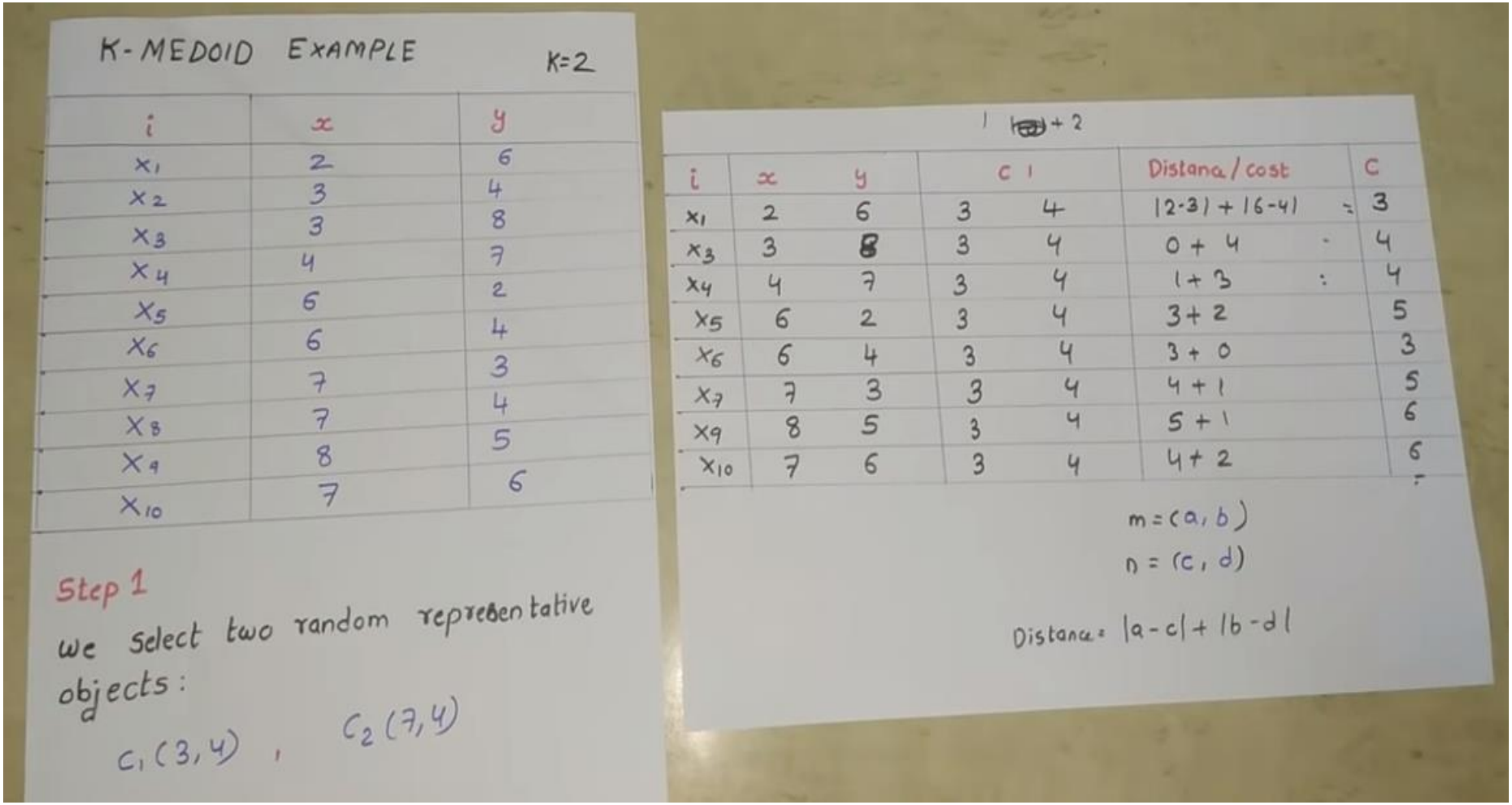
## How is it related to Linear Regression

- KNN has higher prediction power as compare to Linear regression – takes care of the non-linearity
- However, if the linear regression function is close to reality then it will perform better

# K-means clustering algorithms



# K-medoid clustering algorithms



K-MEDOID EXAMPLE K=2

| i | x | y |
|---|---|---|
| $x_1$ | 2 | 6 |
| $x_2$ | 3 | 4 |
| $x_3$ | 3 | 8 |
| $x_4$ | 4 | 7 |
| $x_5$ | 6 | 2 |
| $x_6$ | 6 | 4 |
| $x_7$ | 7 | 3 |
| $x_8$ | 7 | 4 |
| $x_9$ | 8 | 5 |
| $x_{10}$ | 7 | 6 |

Step 1

we select two random representative objects:

$c_1(3,4)$ , $c_2(7,4)$

1 + 2

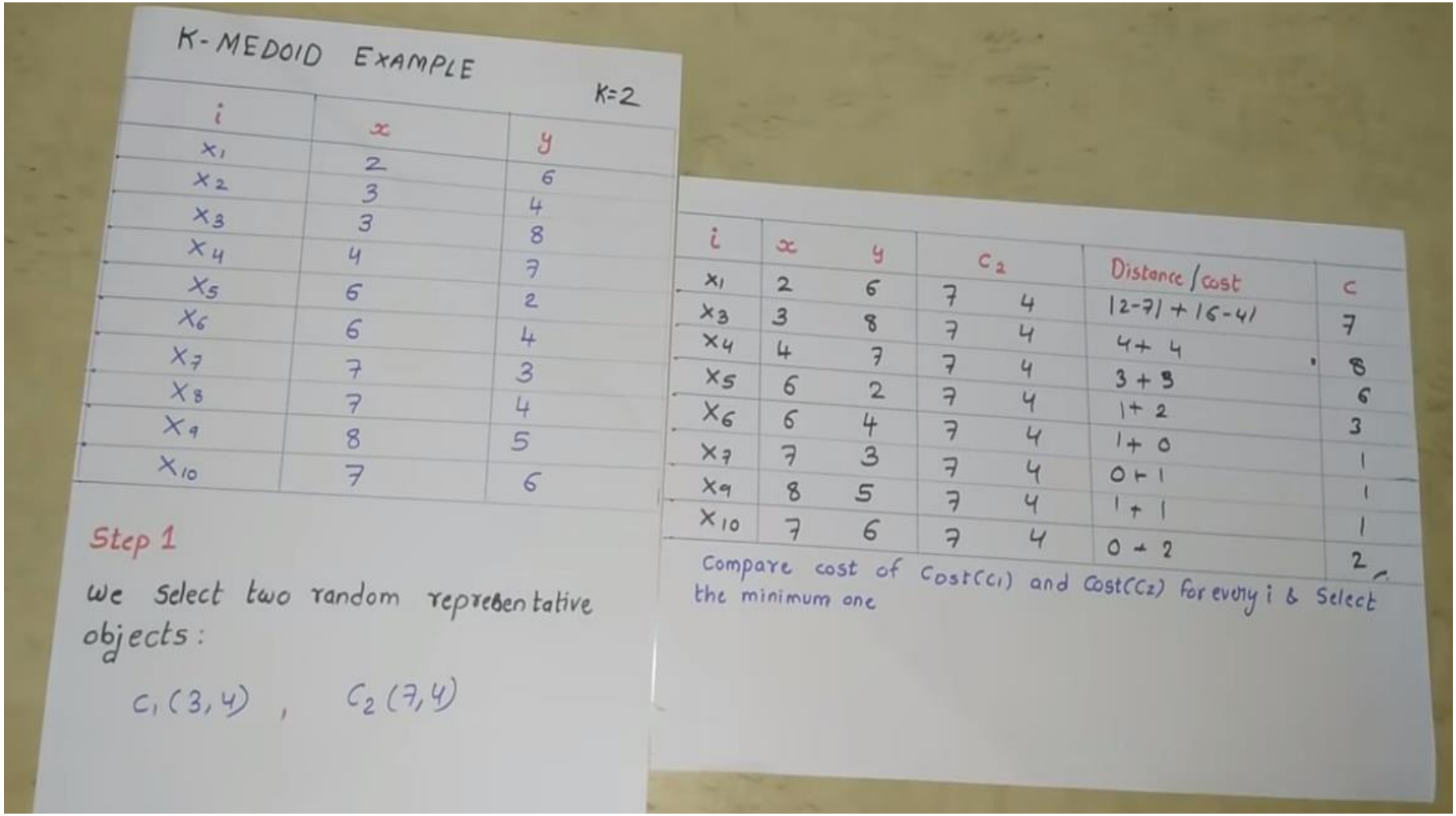
| i | x | y | c 1 | | Distance / cost | | C |
|---|---|---|---|---|---|---|---|
| $x_1$ | 2 | 6 | 3 | 4 | \|2-3\| + \|6-4\| | = | 3 |
| $x_3$ | 3 | 8 | 3 | 4 | 0 + 4 | = | 4 |
| $x_4$ | 4 | 7 | 3 | 4 | 1 + 3 | = | 4 |
| $x_5$ | 6 | 2 | 3 | 4 | 3 + 2 | | 5 |
| $x_6$ | 6 | 4 | 3 | 4 | 3 + 0 | | 3 |
| $x_7$ | 7 | 3 | 3 | 4 | 4 + 1 | | 5 |
| $x_9$ | 8 | 5 | 3 | 4 | 5 + 1 | | 6 |
| $x_{10}$ | 7 | 6 | 3 | 4 | 4 + 2 | | 6 |

m = (a, b)

n = (c, d)

Distance = $|a-c| + |b-d|$

# K-medoid clustering algorithms



K-MEDOID EXAMPLE K=2

| i | x | y |
|---|---|---|
| $x_1$ | 2 | 6 |
| $x_2$ | 3 | 4 |
| $x_3$ | 3 | 8 |
| $x_4$ | 4 | 7 |
| $x_5$ | 6 | 2 |
| $x_6$ | 6 | 4 |
| $x_7$ | 7 | 3 |
| $x_8$ | 7 | 4 |
| $x_9$ | 8 | 5 |
| $x_{10}$ | 7 | 6 |

Step 1

we select two random representative objects:

$c_1(3,4)$ , $c_2(7,4)$

| i | x | y | $c_2$ | | Distance / cost | c |
|---|---|---|---|---|---|---|
| $x_1$ | 2 | 6 | 7 | 4 | $\|2-7\| + \|6-4\|$ | 7 |
| $x_3$ | 3 | 8 | 7 | 4 | 4 + 4 | 8 |
| $x_4$ | 4 | 7 | 7 | 4 | 3 + 3 | 6 |
| $x_5$ | 6 | 2 | 7 | 4 | 1 + 2 | 3 |
| $x_6$ | 6 | 4 | 7 | 4 | 1 + 0 | 1 |
| $x_7$ | 7 | 3 | 7 | 4 | 0 + 1 | 1 |
| $x_9$ | 8 | 5 | 7 | 4 | 1 + 1 | 1 |
| $x_{10}$ | 7 | 6 | 7 | 4 | 0 + 2 | 2 |

Compare cost of Cost($C_1$) and Cost($C_2$) for every i & Select the minimum one

# K-medoid clustering algorithms

| i | x | y | c 1 | | Distance / cost | | C | C |
|---|---|---|---|---|---|---|---|---|
| $x_1$ | 2 | 6 | 3 | 4 | \|2-3\| + \|6-4\| | = | 3 | 7 |
| $x_3$ | 3 | 8 | 3 | 4 | 0 + 4 | - | 4 | 8 |
| $x_4$ | 4 | 7 | 3 | 4 | 1 + 3 | : | 4 | 6 |
| $x_5$ | 6 | 2 | 3 | 4 | 3 + 2 | | 5 | 3 |
| $x_6$ | 6 | 4 | 3 | 4 | 3 + 0 | | 3 | 1 |
| $x_7$ | 7 | 3 | 3 | 4 | 4 + 1 | | 5 | 1 |
| $x_9$ | 8 | 5 | 3 | 4 | 5 + 1 | | 6 | 2 |
| $x_{10}$ | 7 | 6 | 3 | 4 | 4 + 2 | | 6 | 2 |

elect

$m = (a, b)$

$n = (c, d)$

Distance = $|a - c| + |b - d|$

# K-medoid clustering algorithms

Step II) then cluster are

cluster 1: {(2,6), (3,8), (4,7), (3,4)}

cluster 2: {(7,4), (6,2), (6,4), (7,3), (8,5), (7,6)}

Calculate total cost

$$T\ cost(x,c) = \sum_{i=1}^{d} |x_i - c_i|$$

Total cost = {cost((3,4),(2,6)), cost((3,4),(3,8)),
cost((3,4),(4,7)), cost((7,4),(8,5)),
cost((7,4),(6,2)), cost((7,4),(6,4)),
cost((7,4),(7,3)), cost((7,4),(7,6)),

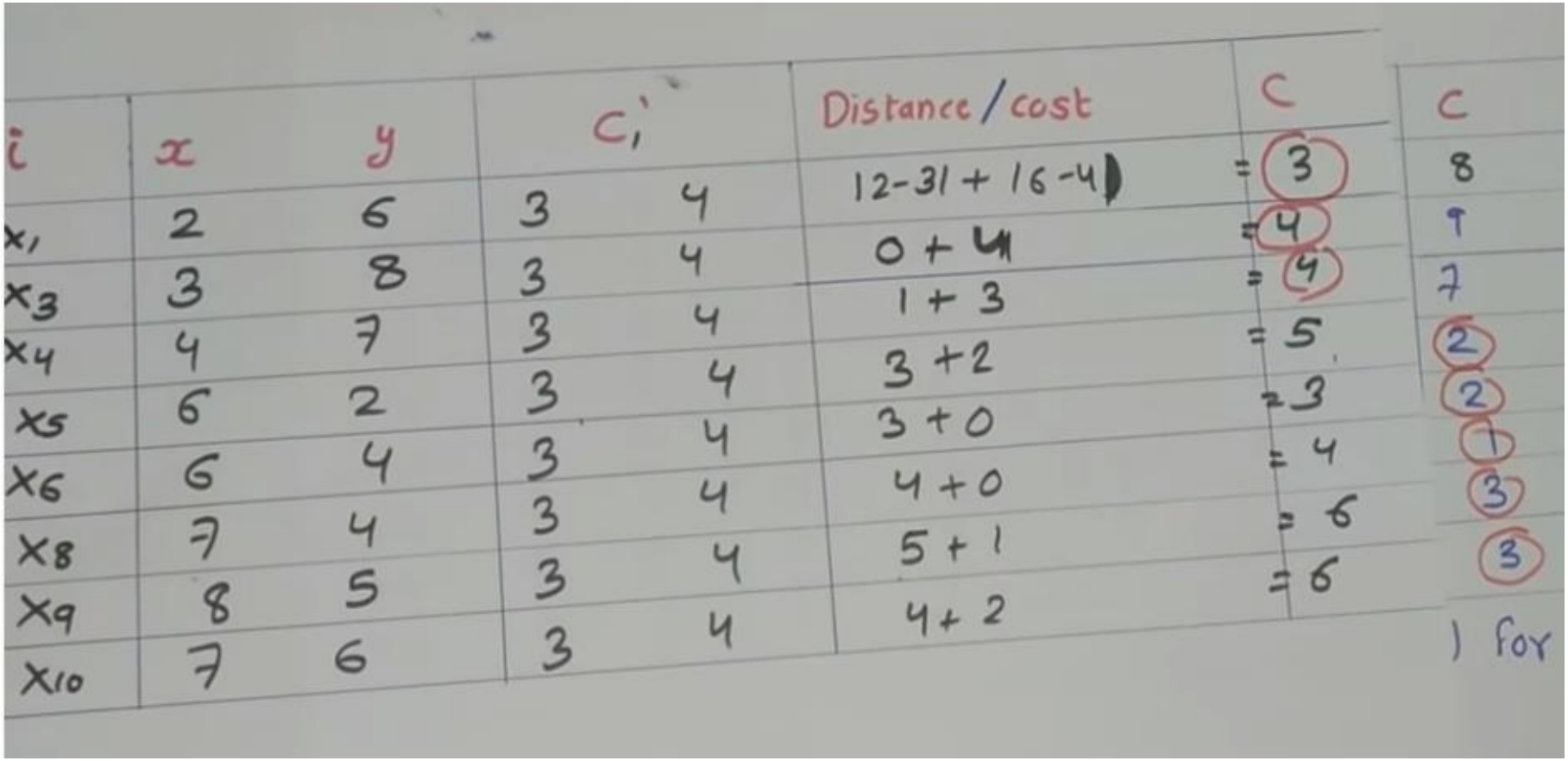
= (3+4+4) + (3+1+1+2+2)

= 20

Step 3) Select one of non-Medoids O'

Let's O' = (7,3) i.e x7

# K-medoid clustering algorithms



| i | x | y | $c_i$ | | Distance / cost | C | C |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| $x_1$ | 2 | 6 | 3 | 4 | 12-31 + 16-4 | = 3 | 8 |
| $x_3$ | 3 | 8 | 3 | 4 | 0 + 4 | = 4 | 9 |
| $x_4$ | 4 | 7 | 3 | 4 | 1 + 3 | = 4 | 7 |
| $x_5$ | 6 | 2 | 3 | 4 | 3 + 2 | = 5 | 2 |
| $x_6$ | 6 | 4 | 3 | 4 | 3 + 0 | = 3 | 2 |
| $x_8$ | 7 | 4 | 3 | 4 | 4 + 0 | = 4 | 1 |
| $x_9$ | 8 | 5 | 3 | 4 | 5 + 1 | = 6 | 3 |
| $x_{10}$ | 7 | 6 | 3 | 4 | 4 + 2 | = 6 | 3 |

) for

# K-medoid clustering algorithms

Again create the cluster

cluster 1 : {(3,4), (2,6), (3,8), (4,7)}

cluster 2 : {(7,3), (6,2), (5,4), (7,4), (8,5), (7,6)}

Current total cost = (3+4+4) + (2+2+1+3+3)

= 11 + 11

= 22

Step 4) So cost of swapping medoid from $C_2$ to O' is

S = current total cost − past total cost

= 22 − 20

= 2 > 0

so, moving O' would be a bad idea so previous choice was Good

# Silhouette algorithm

- K-means is an unsupervised machine learning technique primarily used for the clubbing similar instances together
- In case of supervised learning, there are multiple ways of validating the results
- But in case of unsupervised learning, the only way to validate the score is trough visualization. But this technique also has its limitations which is the higher number of dimensions
- While training the model, an expert is required to decide the number of the clusters that should be there. This becomes very difficult as the dimension of the data increases. There are two simple and popular methods of doing this

  - Inertia: It is defined as the mean squared distance between each instance and its closest centroid
  - Silhouette Score: This is a better measure to decide the number of clusters to be formulated from the data

# Silhouette algorithm

- Silhouette is calculated for each instance and the formula goes like this:

Silhouette Coefficient = (x-y)/ max(x, y)

where,

y is the mean intra cluster distance: mean distance to the other instances in the same cluster

x depicts mean nearest cluster distance i.e. mean distance to the instances of the next closest cluster

- The coefficient varies between -1 and 1. A value close to 1 implies that the instance is close to its cluster is a part of the right cluster. Whereas a value close to -1 means that the value is assigned to the wrong cluster

# Hierarchical Clustering

- Hierarchical clustering is another unsupervised learning algorithm that is used to group together the unlabeled data points having similar characteristics. Hierarchical clustering algorithms falls into following two categories

- Agglomerative hierarchical algorithms− In agglomerative hierarchical algorithms, each data point is treated as a single cluster and then successively merge or agglomerate (bottom-up approach) the pairs of clusters. The hierarchy of the clusters is represented as a dendrogram or tree structure

- Divisive hierarchical algorithms− On the other hand, in divisive hierarchical algorithms, all the data points are treated as one big cluster and the process of clustering involves dividing (Top-down approach) the one big cluster into various small clusters

# Agglomerative hierarchical algorithms

- Step 1 − The number of data points are K at start. Treat each data point as single cluster. Hence, we will have K clusters at start.
- Step 2 − Now, in this step we need to form a big cluster by joining two closet data points. This will result in total of K-1 clusters
- Step 3 − Now, to form more big clusters we need to join two closet clusters. This will result in total of K-2 clusters
- Step 4 − Now, to form one big cluster repeat the above three steps until K would become 0 i.e. no more data points left to join
- Step 5 − At last, after making one single big cluster, dendrograms will be used to divide into multiple clusters depending upon the problem

# Agglomerative hierarchical algorithms

- Agglomerative: Agglomerative is a bottom-up approach, in which the algorithm starts with taking all data points as single clusters and merging them until one cluster is left
- Divisive: Divisive algorithm is the reverse of the agglomerative algorithm as it is a top-down approach

- Divisive clustering : This algorithm also does not require to prespecify the number of clusters. Top-down clustering requires a method for splitting a cluster that contains the whole data and proceeds by splitting clusters recursively until individual data have been splitted into singleton cluster

# Agglomerative vs Divisive clustering

- Divisive clustering is more complex as compared to agglomerative clustering
- Divisive clustering is more efficient if we do not generate a complete hierarchy all the way down to individual data leaves
- Time complexity of a naive agglomerative clustering is $O(n^3)$ because we exhaustively scan the N x N matrix for the lowest distance in each of N-1 iterations. Using priority queue data structure we can reduce this complexity to $O(n^2 \log n)$. By using some more optimizations it can be brought down to $O(n^2)$
- Whereas for divisive clustering given a fixed number of top levels, using an efficient flat algorithm like K-Means, divisive algorithms are linear in the number of patterns and clusters
- Divisive algorithm is also more accurate

# K-D Trees

- A K-D Tree (also called as K-Dimensional Tree) is a binary search tree where data in each node is a K-Dimensional point in space. In short, it is a space partitioning data structure for organizing points in a K-Dimensional space.

- A non-leaf node in K-D tree divides the space into two parts, called as half-spaces.

- Points to the left of this space are represented by the left subtree of that node and points to the right of the space are represented by the right subtree.

- For the sake of simplicity, let us understand a 2-D Tree with an example

# K-D Trees

- The root would have an x-aligned plane, the root's children would both have y-aligned planes, the root's grandchildren would all have x-aligned planes, and the root's great-grandchildren would all have y-aligned planes and so on

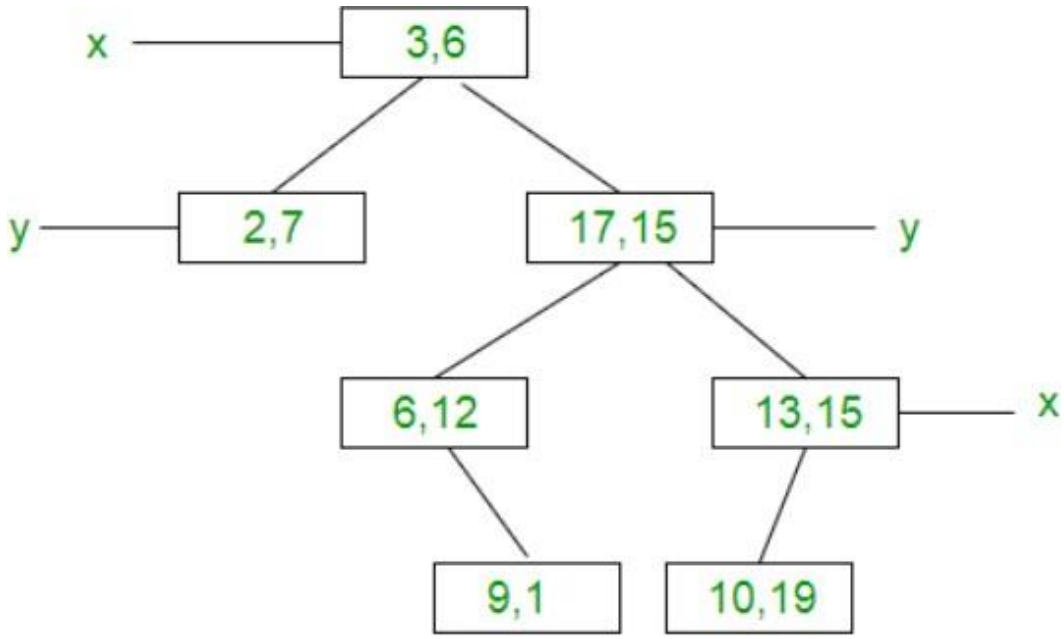
- Example
- Consider following points in a 2-D plane:
- (3, 6), (17, 15), (13, 15), (6, 12), (9, 1), (2, 7), (10, 19)

- Insert (3, 6): Since tree is empty, make it the root node.
- Insert (17, 15): Compare it with root node point. Since root node is X-aligned, the X-coordinate value will be compared to determine if it lies in the left subtree or in the right subtree. This point will be Y-aligned.

# K-D Trees

- Insert (13, 15): X-value of this point is greater than X-value of point in root node. So, this will lie in the right subtree of (3, 6). Again Compare Y-value of this point with the Y-value of point (17, 15) (Why?). Since, they are equal, this point will lie in the right subtree of (17, 15). This point will be X-aligned.
- Insert (6, 12): X-value of this point is greater than X-value of point in root node. So, this will lie in the right subtree of (3, 6). Again Compare Y-value of this point with the Y-value of point (17, 15) (Why?). Since, 12 < 15, this point will lie in the left subtree of (17, 15). This point will be X-aligned.
- Insert (9, 1):Similarly, this point will lie in the right of (6, 12).
- Insert (2, 7):Similarly, this point will lie in the left of (3, 6).
- Insert (10, 19): Similarly, this point will lie in the left of (13, 15).

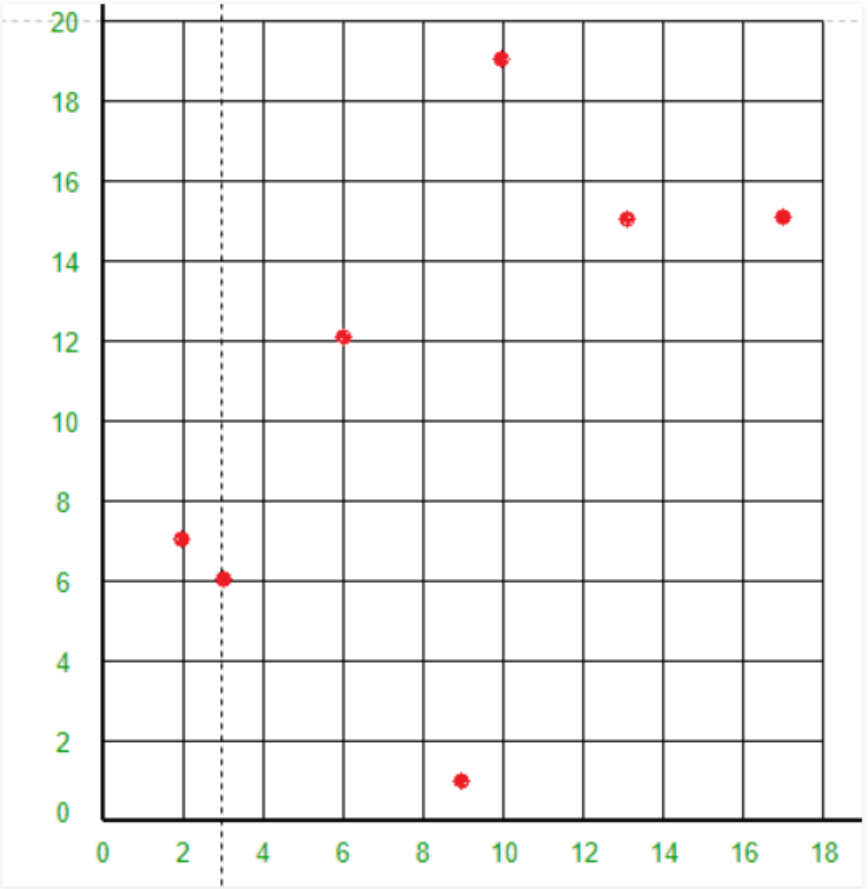
# K-D Trees



# K-D Trees

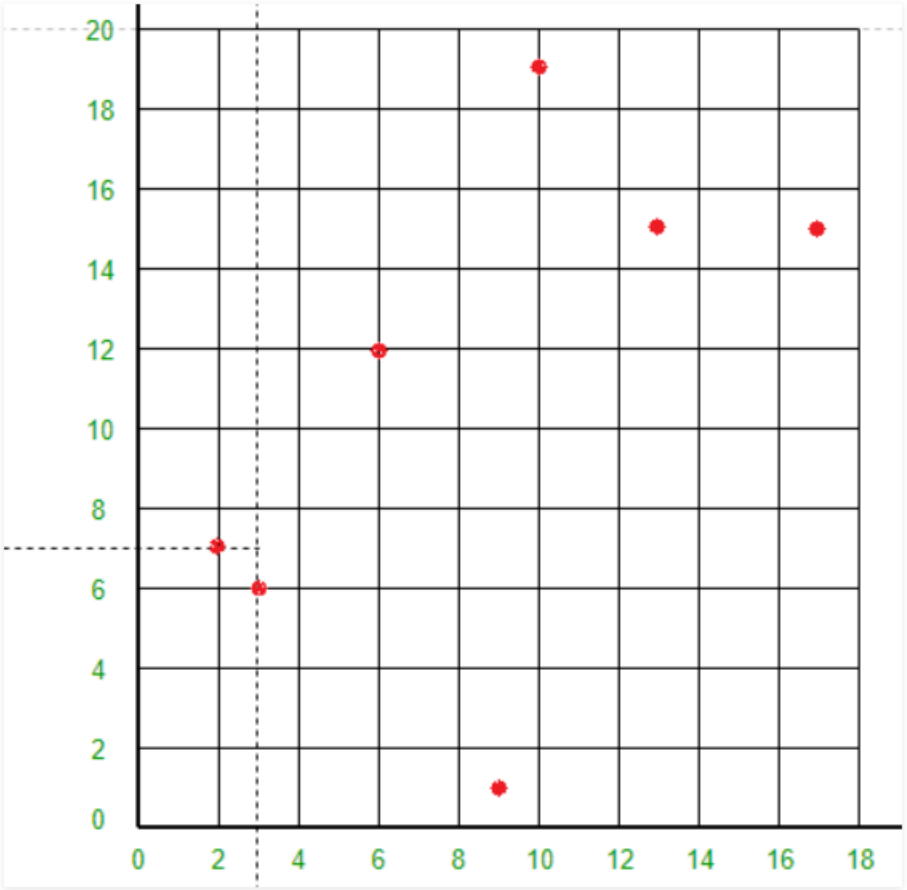
All 7 points will be plotted in the X-Y plane as follows:

1. Point (3, 6) will divide the space into two parts: Draw line X = 3.



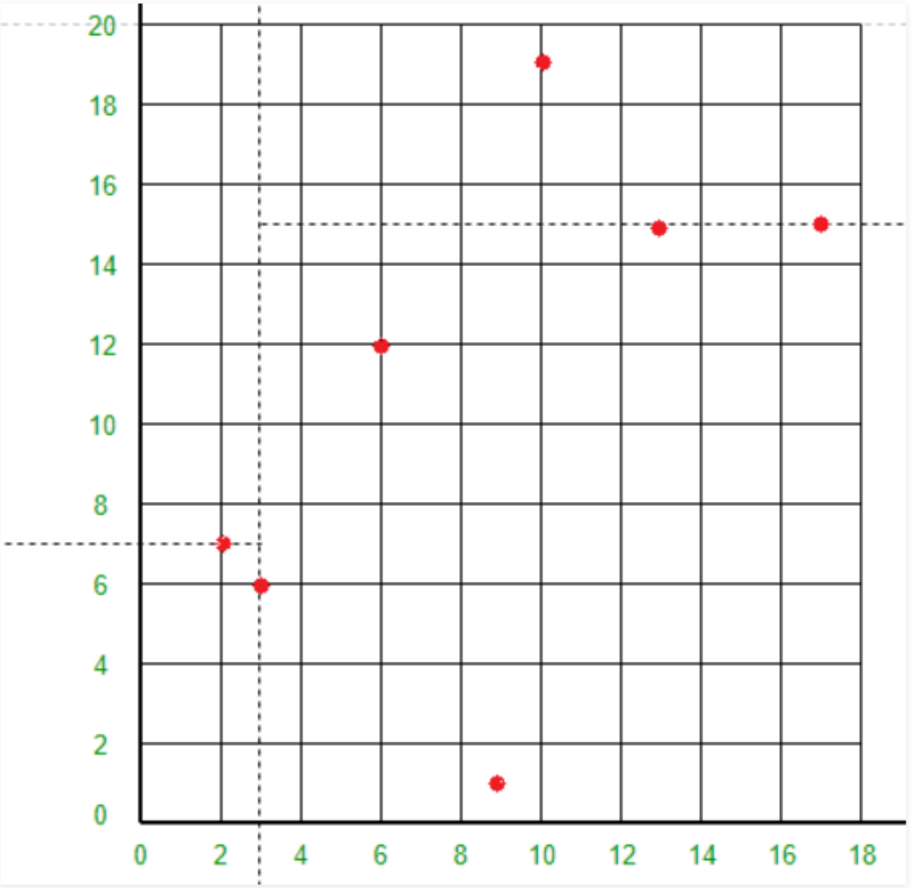
# K-D Trees

2. Point (2, 7) will divide the space to the left of line X = 3 into two parts horizontally. Draw line Y = 7 to the left of line X = 3.



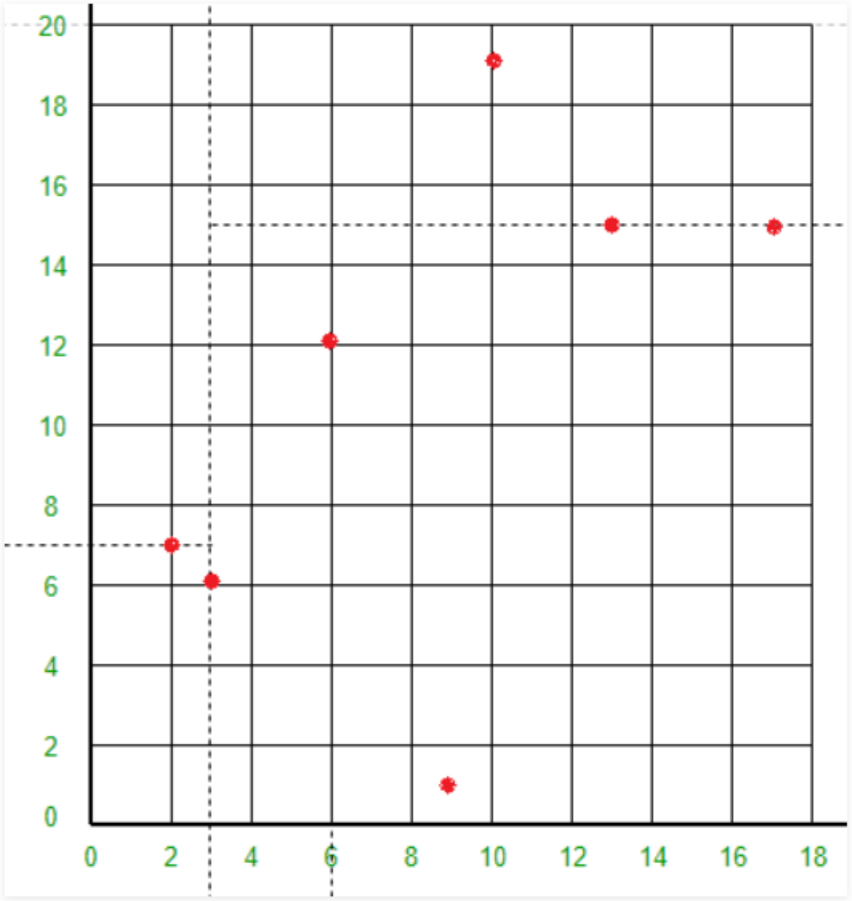
# K-D Trees

3. Point (17, 15) will divide the space to the right of line X = 3 into two parts horizontally. Draw line Y = 15 to the right of line X = 3.



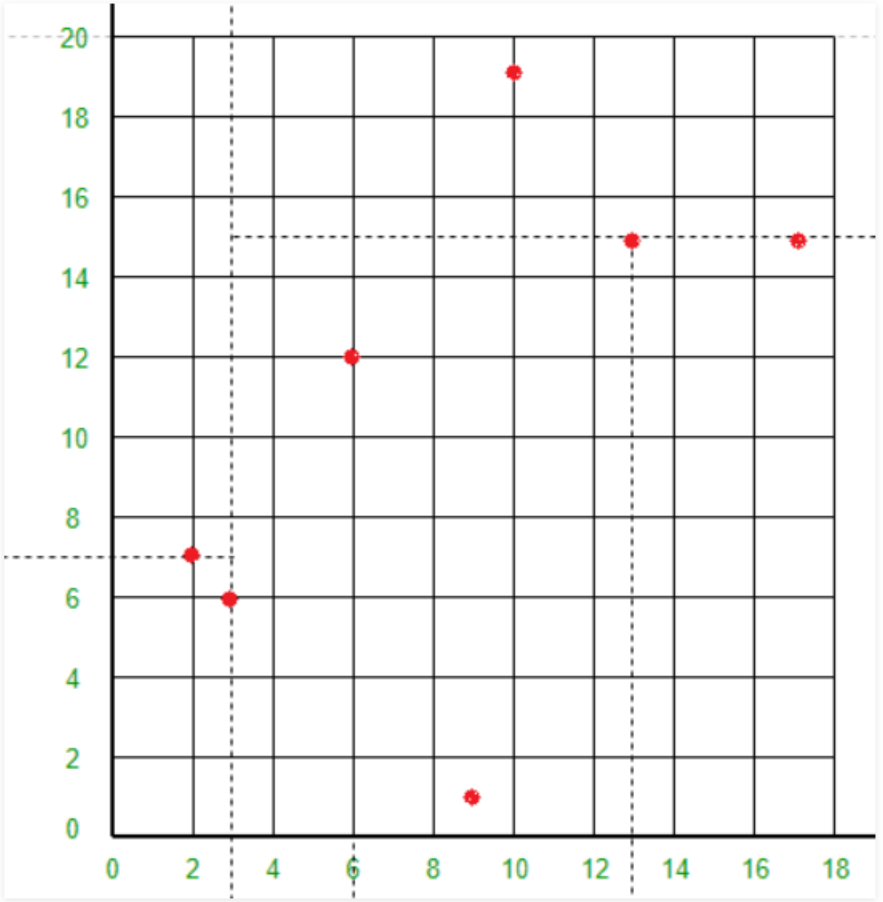
# K-D Trees

- Point (6, 12) will divide the space below line Y = 15 and to the right of line X = 3 into two parts. Draw line X = 6 to the right of line X = 3 and below line Y = 15.



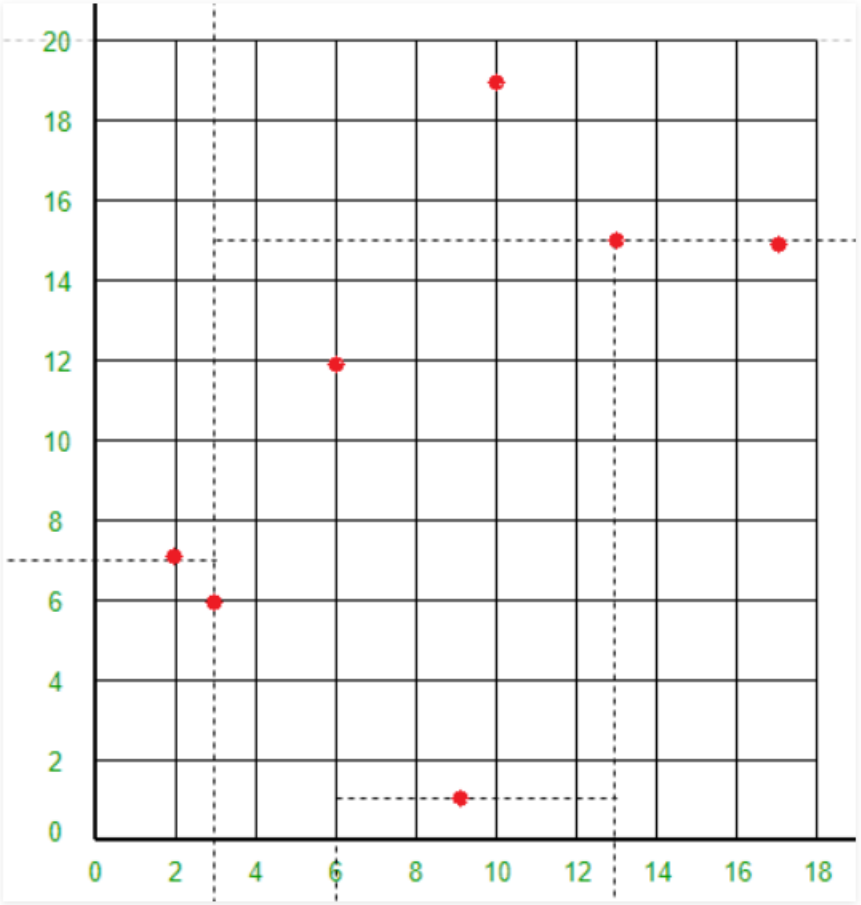
# K-D Trees

- Point (13, 15) will divide the space below line Y = 15 and to the right of line X = 6 into two parts. Draw line X = 13 to the right of line X = 6 and below line Y = 15.



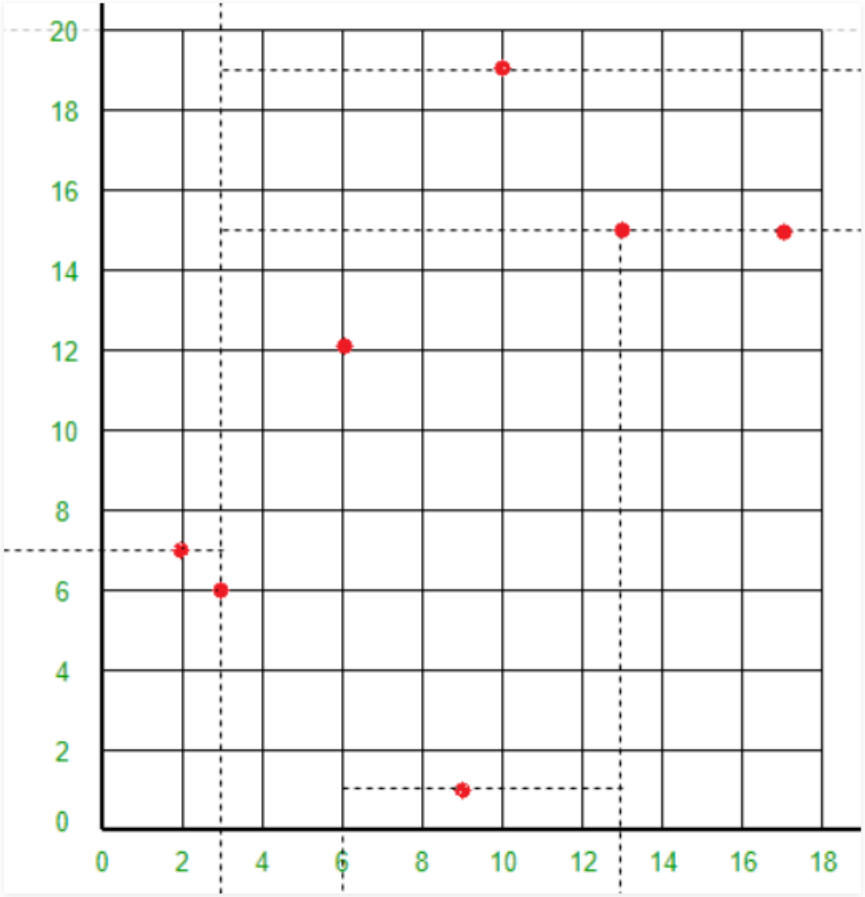
# K-D Trees

- Point (9, 1) will divide the space between lines X = 3, X = 6 and Y = 15 into two parts. Draw line Y = 1 between lines X = 3 and X = 6.



# K-D Trees

- Point (10, 19) will divide the space to the right of line X = 3 and above line Y = 15 into two parts. Draw line Y = 19 to the right of line X = 3 and above line Y = 15.



# Locality Sensitive Hashing

- Locality-sensitive hashing (LSH) is an algorithm that hashes similar input items into the same "buckets" with high probability. Since similar items end up in the same buckets, this algorithm can be used for data clustering and nearest neighbor search. It differs from conventional hashing techniques in that hash collisions are maximized, not minimized

- LSH is a set of techniques that dramatically speed up search-for-neighbors or near-duplication detection on data

- These techniques can be used, for example, to filter out duplicates of scraped web pages at an impressive speed, or to perform near-constant-time lookups of nearby points from a geospatial data set

- It is a method to generate hash codes for data-points with the property that similar data-points will have the same hash codes

# Locality Sensitive Hashing

- Locality-sensitive hash functions are specifically designed so that hash value collisions are more likely for two input values that are close together than for inputs that are far apart

Why an LSH is faster ?

- So far we've been sticking to 2-dimensional data because that's easier to visualize in an article. However, if you think about computing 10 hashes for every 2-dimensional point in order to find neighbors, it may feel like you're doing more work than the simple solution of a linear search through your points

- There are two ways an LSH can speed things up: by helping you deal with a huge number of points, or by helping you deal with points in a high-dimensional space such as image or video data

# Locality Sensitive Hashing

- If you want to find all points close to a query point q, you could execute a full linear search. A single-hash LSH can give you results in constant time. That's faster.

- Things are slightly more complex for higher values of j and k. If you keep j=k, then your LSH result is a simple intersection of k different lists, each list being the set of hash collision points returned by a given randomized hash function hi(). Finding this intersection can be sped up by starting with the smallest of these lists and shrinking it by throwing out points not present in the other lists. This throwing-out process can be done quickly by using hash-table lookups to test for inclusion in the point lists, which are basically constant time. The running time of this approach is essentially O(mk), where m is the length of the shortest list returned by any of your hash functions hi(). This running time is very likely to be an order of magnitude faster than linear search.

# Non-Parametric Regression

- Nonparametric regression is a category of regression analysis in which the predictor does not take a predetermined form but is constructed according to information derived from the data. That is, no parametric form is assumed for the relationship between predictors and dependent variable

- Nonparametric regression requires larger sample sizes than regression based on parametric models because the data must supply the model structure as well as the model estimates

- Nonparametric regression, like linear regression, estimates mean outcomes for a given set of covariates. Unlike linear regression, nonparametric regression is agnostic about the functional form between the outcome and the covariates and is therefore not subject to misspecification error.

# Non-Parametric Regression

In nonparametric regression, you do not specify the functional form. You specify the dependent variable—the outcome—and the covariates. You specify y,x1,x2, and x3 to fit

$$y=g(x1,x2,x3)+\epsilon$$

The method does not assume that g() is linear; it could just as well be

$$y=\beta 1x1+\beta 2x22+\beta 3x31x2+\beta 4x3+\epsilon$$

The method does not even assume the function is linear in the parameters. It could just as well be

$$y=\beta 1x\beta 21+\cos(x2x3)+\epsilon$$

Or it could be anything else.

# Non-Parametric Regression Examples

- nearest neighbors, see nearest-neighbor interpolation and k-nearest neighbors algorithm
- regression trees
- kernel regression
- local regression
- multivariate adaptive regression splines
- neural networks
- support vector regression
- smoothing splines

# Parametric versus Nonparametric Regression

The general linear model is a form of parametric regression, where the relationship between X and Y has some predetermined form.

- Parameterizes relationship between X and Y, e.g., $\hat{Y} = \beta 0 + \beta 1 X$
- Then estimates the specified parameters, e.g., $\beta 0$ and $\beta 1$
- Great if you know the form of the relationship (e.g., linear)

In contrast, nonparametric regression tries to estimate the form of the relationship between X and Y.

- No predetermined form for relationship between X and Y
- Great for discovering relationships and building prediction models

# Ensemble Learning

Ensemble methods are meta-algorithms that combine several machine learning techniques into one predictive model in order to decrease variance (bagging), bias (boosting), or improve predictions (stacking).

Ensemble methods can be divided into two groups:

Sequential ensemble methods where the base learners are generated sequentially (e.g. AdaBoost).

- The basic motivation of sequential methods is to exploit the dependence between the base learners. The overall performance can be boosted by weighing previously mislabeled examples with higher weight

Parallel ensemble methods where the base learners are generated in parallel (e.g. Random Forest).

- The basic motivation of parallel methods is to exploit independence between the base learners since the error can be reduced dramatically by averaging

# Ensemble Learning

- Most ensemble methods use a single base learning algorithm to produce homogeneous base learners, i.e. learners of the same type, leading to homogeneous ensembles

- There are also some methods that use heterogeneous learners, i.e. learners of different types, leading to heterogeneous ensembles. In order for ensemble methods to be more accurate than any of its individual members, the base learners have to be as accurate as possible and as diverse as possible

# Ensemble Learning

## Bagging

- Bagging stands for bootstrap aggregation. One way to reduce the variance of an estimate is to average together multiple estimates
- Bagging uses bootstrap sampling to obtain the data subsets for training the base learners
- we can train M different trees on different subsets of the data and compute the ens

$$f(x) = 1/M \sum_{m=1}^{M} f_m(x)$$

Bagging algorithms: Random Forest

# Ensemble Learning

Random forests: Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model

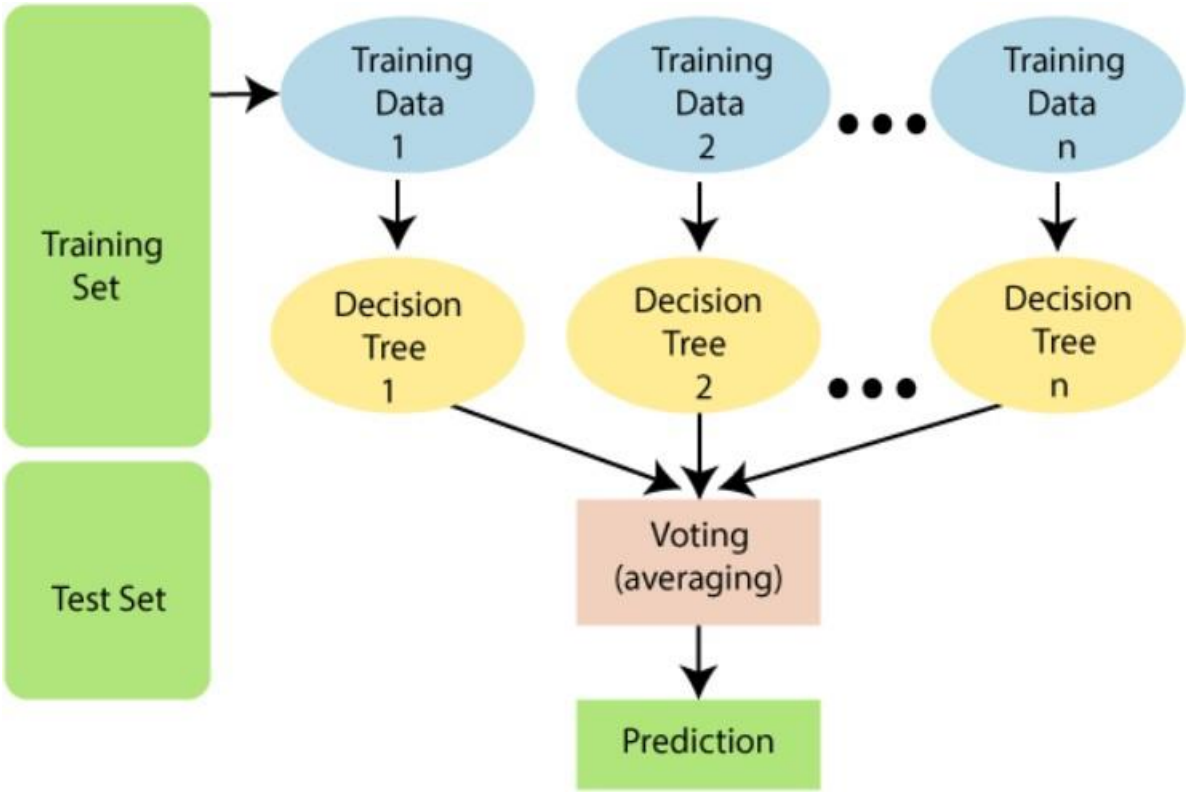
- As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output

- The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting

# Ensemble Learning

Random forests:

The below diagram explains the working of the Random Forest algorithm:



# Ensemble Learning

## Random forests algorithm :

How does Random Forest algorithm work?

Random Forest works in two-phase first is to create the random forest by combining N decision tree, and second is to make predictions for each tree created in the first phase.

Step-1: Select random K data points from the training set.

Step-2: Build the decision tree associated with the selected data points.

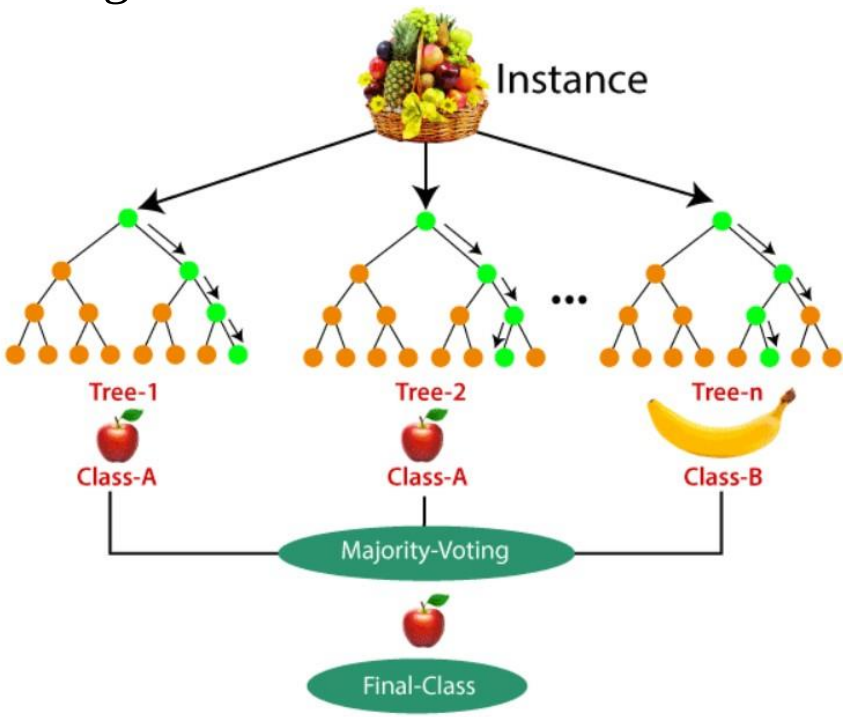
Step-3: Choose number N for number of decision trees that you want to build.

Step-4: Repeat Step 1 & 2 for creation of N decision trees.

Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

# Ensemble Learning

- Example: Suppose there is a dataset that contains multiple fruit images. So, this dataset is given to the Random forest classifier. The dataset is divided into subsets and given to each decision tree. During the training phase, each decision tree produces a prediction result, and when a new data point occurs, then based on the majority of results, the Random Forest classifier predicts the final decision. Consider the below image:



# Ensemble Learning

## Boosting

- Boosting refers to a family of algorithms that can convert weak learners to strong learners. The main principle of boosting is to fit a sequence of weak learners

**Algorithm** AdaBoost

1: Init data weights $\{w_n\}$ to $1/N$
2: **for** $m = 1$ to $M$ **do**
3: fit a classifier $y_m(x)$ by minimizing weighted error function $J_m$:
4: $J_m = \sum_{n=1}^{N} w_n^{(m)} 1[y_m(x_n) \neq t_n]$
5: compute $\epsilon_m = \sum_{n=1}^{N} w_n^{(m)} 1[y_m(x_n) \neq t_n] / \sum_{n=1}^{N} w_n^{(m)}$
6: evaluate $\alpha_m = \log\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$
7: update the data weights: $w_n^{(m+1)} = w_n^{(m)} \exp\{\alpha_m 1[y_m(x_n) \neq t_n]\}$
8: **end for**
9: Make predictions using the final model: $Y_M(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(x)\right)$

# Ensemble Learning

## Stacking

- Stacking is an ensemble learning technique that combines multiple classification or regression models via a meta-classifier or a meta-regressor

**Algorithm** Stacking

1: Input: training data $D = \{x_i, y_i\}_{i=1}^{m}$
2: Ouput: ensemble classifier $H$
3: *Step 1: learn base-level classifiers*
4: **for** $t = 1$ to $T$ **do**
5: learn $h_t$ based on $D$
6: **end for**
7: *Step 2: construct new data set of predictions*
8: **for** $i = 1$ to $m$ **do**
9: $D_h = \{x'_i, y_i\}$, where $x'_i = \{h_1(x_i), ..., h_T(x_i)\}$
10: **end for**
11: *Step 3: learn a meta-classifier*
12: learn $H$ based on $D_h$
13: return $H$

# Meta Learning

- Meta-learning, also known as "learning to learn", intends to design models that can learn new skills or adapt to new environments rapidly with a few training examples

- In practice, very closely related to multi-task learning

- Common approach to meta learning:

Model-Based

  - Memory-Augmented Neural Networks
  - Meta Networks

Metric-Based

  - Convolutional Siamese Neural Network
  - Matching Networks
  - Relation Network
  - Prototypical Networks

Optimization-Based

  - LSTM Meta-Learner
  - Temporal Discreteness
  - Reptile

# Meta Learning

## Model-Based

Model-based meta-learning models updates its parameters rapidly with a few training steps, which can be achieved by its internal architecture or controlled by another meta-learner mode

- Memory-Augmented Neural Networks: The model is known as MANN short for Memory-Augmented Neural Networks, which is expected to encode new information fast and thus to adapt to new tasks after only a few samples, it fits well for meta-learning

- Meta Networks: Meta Networks (MetaNet) learns a meta-level knowledge across tasks and shifts its inductive biases via fast parameterization for rapid generalization

# Meta Learning

## Metric-Based

The core idea in metric-based meta-learning is similar to nearest neighbors algorithms, which weight is generated by a kernel function. It aims to learn a metric or distance function over objects. The notion of a good metric is problem-dependent. It should represent the relationship between inputs in the task space and facilitate problem solving

- Convolutional Siamese Neural Network: Siamese neural network is composed of two twin networks whose output is jointly trained. There is a function above to learn the relationship between input data sample pairs. The two networks are the same, sharing the same weight and network parameters

- Matching Networks: Matching Networks learn a network that maps a small labelled support set and an unlabelled example to its label, obviating the need for fine-tuning to adapt to new class types

# Meta Learning

## Metric-Based

- Relation Network: The Relation Network (RN), is trained end-to-end from scratch. During meta-learning, it learns to learn a deep distance metric to compare a small number of images within episodes, each of which is designed to simulate the few-shot setting

- Prototypical Networks: Prototypical Networks learn a metric space in which classification can be performed by computing distances to prototype representations of each class. Compared to recent approaches for few-shot learning, they reflect a simpler inductive bias that is beneficial in this limited-data regime, and achieve satisfied results

# Meta Learning

Optimization-Based:

What optimization-based meta-learning algorithms intend for is to adjust the optimization algorithm so that the model can be good at learning with a few examples

- LSTM Meta-Learner: LSTM-based meta-learner is to learn the exact optimization algorithm used to train another learner neural network classifier in the few-shot regime. The parametrization allows it to learn appropriate parameter updates specifically for the scenario where a set amount of updates will be made, while also learning a general initialization of the learner (classifier) network that allows for quick convergence of training

- Temporal Discreteness: MAML, short for Model-Agnostic Meta-Learning, is a fairly general optimization algorithm, compatible with any model that learns through gradient descent

# Meta Learning

## Optimization-Based

- Reptile: Reptile is a remarkably simple meta-learning optimization algorithm, given that both rely on meta-optimization through gradient descent and both are model-agnostic

# Meta Learning

Why is meta-learning a good idea?

- Deep reinforcement learning, especially model-free, requires a huge number of samples
- If we can meta-learn a faster reinforcement learner, we can learn new tasks efficiently
- What can a meta-learned learner do differently?
  - Explore more intelligently
  - Avoid trying actions that are know to be useless
  - Acquire the right features more quickly

# Meta Learning

## Meta-learning summary & open problems

- The promise of meta-learning: use past experience to simply acquire a much more efficient deep RL algorithm
- The reality of meta learning: mostly works well on smaller problems but getting better all the time
- Main limitations
  - RNN policies are extremely hard to train, and likely not scalable
  - Model-agnostic meta-learning presents a tough optimization problem
  - Designing the right task distribution is hard
  - Generally very sensitive to task distribution (meta-overfitting)


THANK YOU