## 5.1 Non parametric classification

- Machine learning algorithms are classified as two distinct groups: parametric and non-parametric. In parametricness is related to pair of model complexity and the number of rows in the train set.
- We can classify algorithms as non-parametric when model becomes more complex if number of samples in the training set increases. Vice versa, a model would be parametric if model becomes stable when number of examples in the training set increases.
  - **Parametric Machine Learning Algorithms**
- Assumptions can greatly simplify the learning process, but can also limit what can be learned. Algorithms that simplify the function to a known form are called parametric machine learning algorithms.

A learning model that summarizes data with a set of parameters of fixed size (independent of the number of training examples) is called a parametric model. No matter how much data you throw at a parametric model, it won't change its mind about how many parameters it needs.

The algorithms involve two steps:

- Select a form for the function.
- Learn the coefficients for the function from the training data.

An easy to understand functional form for the mapping function is a line, as is used in linear regression:

$$b0 + b1*x1 + b2*x2 = 0$$

- Where b0, b1 and b2 are the coefficients of the line that control the intercept and slope, and x1 and x2 are two input variables.
- Assuming the functional form of a line greatly simplifies the learning process. Now, all we need to do is estimate the coefficients of the line equation and we have a predictive model for the problem.

Some more examples of parametric machine learning algorithms include:

- Logistic Regression
- Naive Bayes
- Simple Neural Networks

**Benefits of Parametric Machine Learning Algorithms:**

- **Simpler**: These methods are easier to understand and interpret results.
- **Speed**: Parametric models are very fast to learn from data.

- **Less Data**: They do not require as much training data and can work well even if the fit to the data is not perfect.

**Limitations of Parametric Machine Learning Algorithms:**

- **Constrained**: By choosing a functional form these methods are highly constrained to the specified form.
- **Limited Complexity**: The methods are more suited to simpler problems.
- **Poor Fit**: In practice the methods are unlikely to match the underlying mapping function

**Nonparametric Machine Learning Algorithms**.

- Algorithms that do not make strong assumptions about the form of the mapping function are called nonparametric machine learning algorithms. By not making assumptions, they are free to learn any functional form from the training data.
- Nonparametric methods are good when you have a lot of data and no prior knowledge, and when you don't want to worry too much about choosing just the right features.
- Nonparametric methods seek to best fit the training data in constructing the mapping function, whilst maintaining some ability to generalize to unseen data. As such, they are able to fit a large number of functional forms.
- An easy to understand nonparametric model is the k-nearest neighbors algorithm that makes predictions based on the k most similar training patterns for a new data instance. The method does not assume anything about the form of the mapping function other than patterns that are close are likely to have a similar output variable.

Some more examples of popular nonparametric machine learning algorithms are:

- k-Nearest Neighbors
- Decision Trees like CART, ID3 and C4.5
- Support Vector Machines

**Benefits of Nonparametric Machine Learning Algorithms:**

- **Flexibility**: Capable of fitting a large number of functional forms.
- **Power**: No assumptions (or weak assumptions) about the underlying function.
- **Performance**: Can result in higher performance models for prediction.

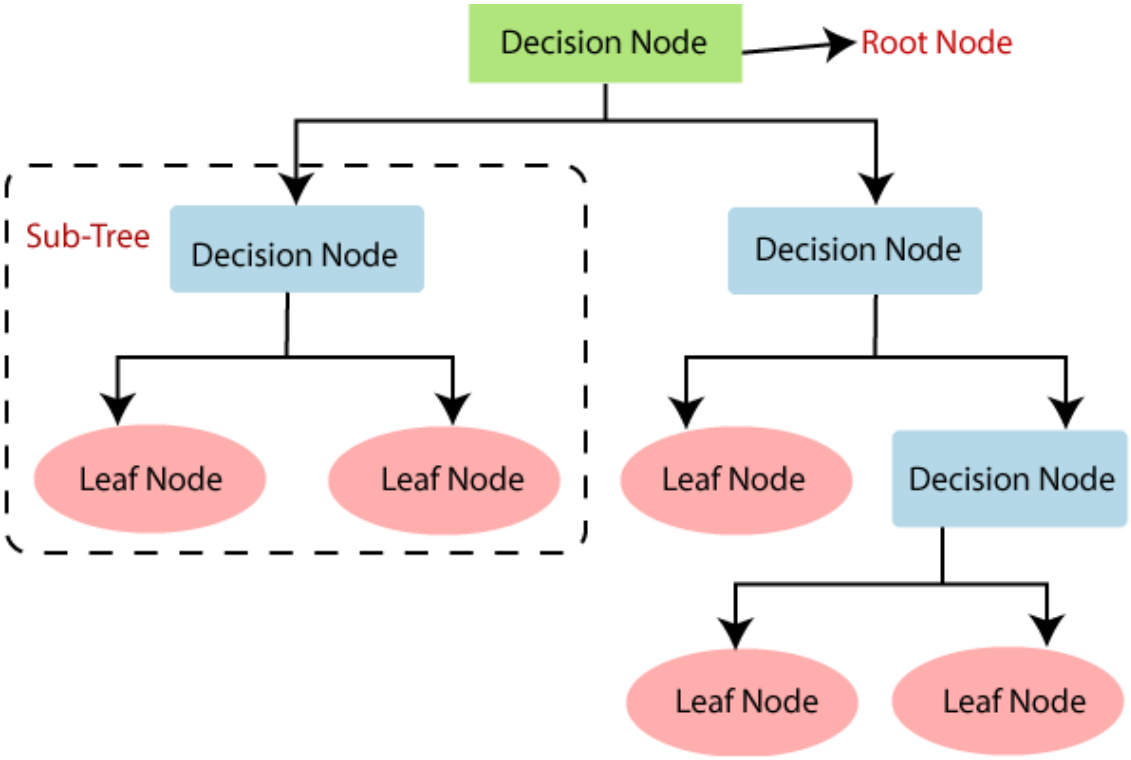
**Limitations of Nonparametric Machine Learning Algorithms:**

- **More data**: Require a lot more training data to estimate the mapping function.
- **Slower**: A lot slower to train as they often have far more parameters to train.
- **Overfitting**: More of a risk to overfit the training data and it is harder to explain why specific predictions are made.

## 5.1.2 Decision Trees: Entropy, Information Gain, Splitting criteria.

## Decision Tree Algorithm

- Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems.
- It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome.**
- In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node.**
- Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
- **It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.**
- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
- In order to build a tree, we use the **CART algorithm,** which stands for **Classification and Regression Tree algorithm.**
- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.
- The general structure of a decision tree




- **Root Node:** Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.
- **Leaf Node:** Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
- **Splitting:** Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
- **Branch/Sub Tree:** A tree formed by splitting the tree.
- **Pruning:** Pruning is the process of removing the unwanted branches from the tree.
- **Parent/Child node:** The root node of the tree is called the parent node, and other nodes are called the child nodes.

## Why use Decision Trees?

- There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model. Below are the two reasons for using the Decision tree:
- Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
- The logic behind the decision tree can be easily understood because it shows a tree-like structure.

## How does the Decision Tree algorithm Work?

- In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.
- For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:
- **Step-1:** Begin the tree with the root node, says S, which contains the complete dataset. Find the best attribute in the dataset using **Attribute Selection Measure (ASM).**
- **Step-2:** Divide the S into subsets that contains possible values for the best attributes.
- **Step-3:** Recursively make new decision trees using the subsets of the dataset created in step -2. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

## Algorithms

- **CART (Gini Index)**
- **ID3 (Entropy, Information Gain)**
- While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM.** By this measurement, we can easily select the best attribute for the nodes of the tree.
- There are two popular techniques for ASM, which are:
    - **Information Gain**

- **Gini Index**

## Information Gain

- Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.
- It calculates how much information a feature provides us about a class.
- According to the value of information gain, we split the node and build the decision tree.
- A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:
- Information Gain= Entropy(S)-[(Weighted Avg) *Entropy(each feature)
- **Entropy:** Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:
- $\text{Entropy(s)}= -P(\text{yes})\log_2 P(\text{yes})- P(\text{no}) \log_2 P(\text{no})$

## Gini Index

- Gini index is a measure of impurity or purity used while creating a decision tree in the CART (Classification and Regression Tree) algorithm.
- An attribute with the low Gini index should be preferred as compared to the high Gini index.
- It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.
- Gini index can be calculated using the below formula:

$$\text{Gini Index}= 1- \sum_j P_j^2$$

## Pruning

- Pruning: Getting an Optimal Decision tree

- Pruning is a process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree.
- A too-large tree increases the risk of overfitting, and a small tree may not capture all the important features of the dataset. Therefore, a technique that decreases the size of the learning tree without reducing accuracy is known as Pruning. There are mainly two types of tree **pruning** technology used:
    - Cost Complexity Pruning
    - Reduced Error Pruning**.**

## ID3 ALGORITHM

- ID3 stands for Iterative Dichotomiser 3 and is named such because the algorithm iteratively (repeatedly) dichotomizes(divides) features into two or more groups at each step.
- Invented by Ross Quinlan, ID3 uses a **top-down greedy** approach to build a decision tree. In simple words, the **top-down** approach means that we start building the tree from the top and the **greedy** approach means that at each iteration we select the best feature at the present moment to create a node.
- Most generally ID3 is only used for classification problems with nominal features only.
- ID3 algorithm selects the best feature at each step while building a Decision tree.
  How does ID3 select the best feature?
- ID3 uses **Information Gain** or just **Gain** to find the best feature.
- Information Gain calculates the reduction in the entropy and measures how well a given feature separates or classifies the target classes. The feature with the **highest Information Gain** is selected as the **best** one
- In simple words, **Entropy** is the measure of disorder and the Entropy of a dataset is the measure of disorder in the target feature of the dataset.
  In the case of binary classification (where the target column has only two types of classes) entropy is **0** if all values in the target column are homogenous(similar) and will be **1** if the target column has equal number values for both the classes.

EXAMPLE:

## *PlayTennis*: training examples

| Day | Outlook | Temperature | Humidity | Wind | PlayTennis |
|---|---|---|---|---|---|
| D1 | Sunny | Hot | High | Weak | No |
| D2 | Sunny | Hot | High | Strong | No |
| D3 | Overcast | Hot | High | Weak | Yes |
| D4 | Rain | Mild | High | Weak | Yes |
| D5 | Rain | Cool | Normal | Weak | Yes |
| D6 | Rain | Cool | Normal | Strong | No |
| D7 | Overcast | Cool | Normal | Strong | Yes |
| D8 | Sunny | Mild | High | Weak | No |
| D9 | Sunny | Cool | Normal | Weak | Yes |
| D10 | Rain | Mild | Normal | Weak | Yes |
| D11 | Sunny | Mild | Normal | Strong | Yes |
| D12 | Overcast | Mild | High | Strong | Yes |
| D13 | Overcast | Hot | Normal | Weak | Yes |
| D14 | Rain | Mild | High | Strong | No |

1. COMPUTE THE **ENTROPY** FOR DATA-SET **ENTROPY(S)**

2. FOR EVERY ATTRIBUTE/FEATURE:

    1. CALCULATE ENTROPY FOR ALL OTHER VALUES **ENTROPY(A)**

    2. TAKE **AVERAGE INFORMATION ENTROPY** FOR THE CURRENT ATTRIBUTE

    3. CALCULATE **GAIN** FOR THE CURRENT ATTRIBUTE

3. PICK THE **HIGHEST GAIN ATTRIBUTE**.

4. **REPEAT** UNTIL WE GET THE TREE WE DESIRED.

# STEP 1: CREATE A ROOT NODE

- HOW TO CHOOSE THE ROOT NODE?

  The attribute that best classifies the training data, use this attribute at the root of the tree.

- HOW TO CHOOSE THE BEST ATTRIBUTE?

  So from here, *ID3 algorithm* begins

- Calculate **Entropy** (Amount of uncertainity in dataset):

$$Entropy = \frac{-p}{p+n} log_2(\frac{p}{p+n}) - \frac{n}{p+n} log_2(\frac{n}{p+n})$$

- Calculate **Average Information**:

$$I(Attribute) = \sum \frac{p_i + n_i}{p+n} Entropy(A)$$

- Calculate **Information Gain**: (Difference in Entropy before and after splitting dataset on attribute A)

$$Gain = Entropy(S) - I(Attribute)$$

| S. No. | Outlook | Temperature | Humidity | Windy | PlayTennis |
|---|---|---|---|---|---|
| 1 | Sunny | Hot | High | Weak | No |
| 2 | Sunny | Hot | High | Strong | No |
| 3 | Overcast | Hot | High | Weak | Yes |
| 4 | Rainy | Mild | High | Weak | Yes |
| 5 | Rainy | Cool | Normal | Weak | Yes |
| 6 | Rainy | Cool | Normal | Strong | No |
| 7 | Overcast | Cool | Normal | Strong | Yes |
| 8 | Sunny | Mild | High | Weak | No |
| 9 | Sunny | Cool | Normal | Weak | Yes |
| 10 | Rainy | Mild | Normal | Weak | Yes |
| 11 | Sunny | Mild | Normal | Strong | Yes |
| 12 | Overcast | Mild | High | Strong | Yes |
| 13 | Overcast | Hot | Normal | Weak | Yes |
| 14 | Rainy | Mild | High | Strong | No |

P = 9

N = 5

Total = 14

- Calculate **Entropy(S):**

$$Entropy = \frac{-p}{p+n} log_2(\frac{p}{p+n}) - \frac{n}{p+n} log_2(\frac{n}{p+n})$$

$$Entropy(S) = \frac{-9}{9+5} log_2(\frac{9}{9+5}) - \frac{5}{9+5} log_2(\frac{5}{9+5})$$

$$Entropy(S) = \frac{-9}{14} log_2(\frac{9}{14}) - \frac{5}{14} log_2(\frac{5}{14}) \quad = 0.940$$

- For each Attribute: (let say **Outlook**)
  - Calculate Entropy for each Values, i.e for 'Sunny', 'Rainy','Overcast'

| Outlook | PlayTennis |
|---|---|
| Sunny | No |
| Sunny | No |
| Sunny | No |
| Sunny | Yes |
| Sunny | Yes |

| Outlook | PlayTennis |
|---|---|
| Rainy | Yes |
| Rainy | Yes |
| Rainy | No |
| Rainy | Yes |
| Rainy | No |

| Outlook | PlayTennis |
|---|---|
| Overcast | Yes |
| Overcast | Yes |
| Overcast | Yes |
| Overcast | Yes |

| Outlook | p | n | Entropy |
|---|---|---|---|
| Sunny | 2 | 3 | 0.971 |
| Rainy | 3 | 2 | 0.971 |
| Overcast | 4 | 0 | 0 |

- Calculate **Average Information Entropy**:

$$I(Outlook) = \frac{p_{sunny} + n_{sunny}}{p+n} Entropy(Outlook = Sunny) +$$

$$\frac{p_{rainy} + n_{rainy}}{p+n} Entropy(Outlook = Rainy) +$$

$$\frac{p_{Overcast} + n_{Overcast}}{p+n} Entropy(Outlook = Overcast)$$

$$I(Outlook) = \frac{3+2}{9+5} * 0.971 + \frac{2+3}{9+5} * 0.971 + \frac{4+0}{9+5} * 0 \quad = 0.693$$

- Calculate **Gain**: attribute is Outlook

$$\boldsymbol{Gain = Entropy(S) - I(Attribute)}$$

$$\boldsymbol{Entropy(S) = 0.940}$$

$$\boldsymbol{Gain(Outlook) = 0.940 - 0.693 = 0.247}$$

- For each Attribute: (let say **Temperature**)
  - Calculate Entropy for each Temp, i.e for 'Hot', 'Mild' and 'Cool'

| Temperature | PlayTennis |
|---|---|
| Hot | No |
| Hot | No |
| Hot | Yes |
| Hot | Yes |

| Temperature | PlayTennis |
|---|---|
| Mild | Yes |
| Mild | No |
| Mild | Yes |
| Mild | Yes |
| Mild | Yes |
| Mild | No |

| Temperature | PlayTennis |
|---|---|
| Cool | Yes |
| Cool | No |
| Cool | Yes |
| Cool | Yes |

| Temperature | p | n | Entropy |
|---|---|---|---|
| Hot | 2 | 2 | 1 |
| Mild | 4 | 2 | 0.918 |
| Cool | 3 | 1 | 0.811 |

- Calculate **Average Information Entropy**:

$$I(Temperature) = \frac{p_{hot} + n_{hot}}{p+n} Entropy(Temperature = Hot) +$$

$$\frac{p_{mild} + n_{mild}}{p+n} Entropy(Temperature = Mild) +$$

$$\frac{p_{Cool} + n_{Cool}}{p+n} Entropy(Temperature = Cool)$$

$$I(Temperature) = \frac{2+2}{9+5} * 1 + \frac{4+2}{9+5} * 0.918 + \frac{3+1}{9+5} * 0.811 => 0.911$$

- Calculate **Gain**: attribute is Temperature

$$Gain = Entropy(S) - I(Attribute)$$

$$Entropy(S) = 0.940$$

$$Gain(Temperature) = 0.940 - 0.911 = 0.029$$

- For each Attribute: (let say **Humidity**)
  - Calculate Entropy for each Humidity, i.e for 'High', 'Normal'

| Humidity | PlayTennis |
|---|---|
| Normal | Yes |
| Normal | No |
| Normal | Yes |
| Normal | Yes |
| Normal | Yes |
| Normal | Yes |
| Normal | Yes |

| Humidity | PlayTennis |
|---|---|
| High | No |
| High | No |
| High | Yes |
| High | Yes |
| High | No |
| High | Yes |
| High | No |

| Humidity | p | n | Entropy |
|---|---|---|---|
| High | 3 | 4 | 0.985 |
| Normal | 6 | 1 | 0.591 |

---

- Calculate **Average Information Entropy**:

$$I(Humidity) = \frac{p_{High} + n_{High}}{p+n} Entropy(Humidity = High) +$$

$$\frac{p_{Normal} + n_{Normal}}{p+n} Entropy(Humidity = Normal)$$

$$I(Humidity) = \frac{3+4}{9+5} * 0.985 + \frac{6+1}{9+5} * 0.591 => 0.788$$

- For each Attribute: (let say **Windy**)
  - Calculate Entropy for each Windy, i.e for 'Strong' and 'Weak'

| Windy | PlayTennis |
|---|---|
| Weak | No |
| Weak | Yes |
| Weak | Yes |
| Weak | Yes |
| Weak | No |
| Weak | Yes |
| Weak | Yes |
| Weak | Yes |

| Windy | PlayTennis |
|---|---|
| Strong | No |
| Strong | No |
| Strong | Yes |
| Strong | Yes |
| Strong | Yes |
| Strong | No |

| Windy | p | n | Entropy |
|---|---|---|---|
| Strong | 3 | 3 | 1 |
| Weak | 6 | 2 | 0.811 |

- Calculate **Average Information Entropy**:

$$I(Windy) = \frac{p_{Strong} + n_{Strong}}{p+n} Entropy(Windy = Strong) + \frac{p_{Weak} + n_{Weak}}{p+n} Entropy(Windy = Weak)$$

$$I(Windy) = \frac{3+3}{9+5} * 1 + \frac{6+2}{9+5} * 0.811 => 0.892$$

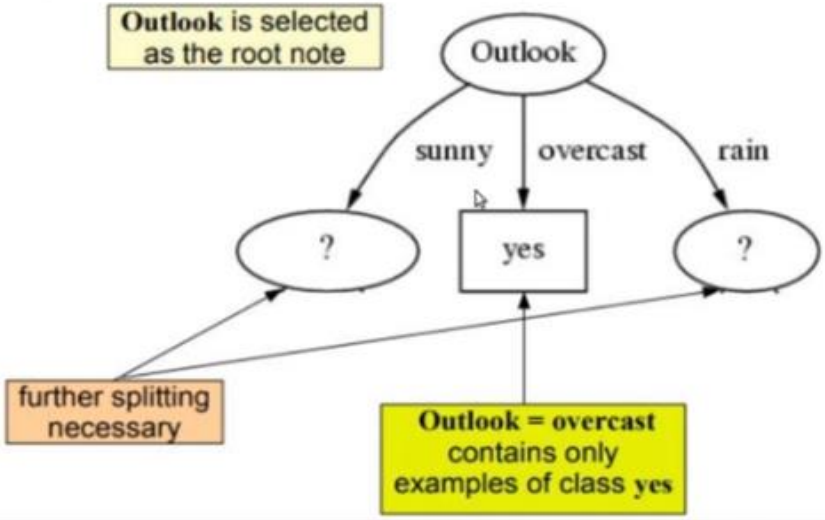
- PICK THE HIGHEST GAIN ATTRIBUTE.

| Attributes | Gain |
|---|---|
| Outlook | 0.247 |
| Temperature | 0.029 |
| Humidity | 0.152 |
| Windy | 0.048 |

ROOT NODE:

**OUTLOOK**

| Outlook | Temperature | Humidity | Windy | PlayTennis |
|---|---|---|---|---|
| Overcast | Hot | High | Weak | Yes |
| Overcast | Cool | Normal | Strong | Yes |
| Overcast | Mild | High | Strong | Yes |
| Overcast | Hot | Normal | Weak | Yes |




- REPEAT THE SAME THING FOR SUB-TREES TILL WE GET THE TREE.

| Outlook | Temperature | Humidity | Windy | PlayTennis |
|---|---|---|---|---|
| Sunny | Hot | High | Weak | No |
| Sunny | Hot | High | Strong | No |
| Sunny | Mild | High | Weak | No |
| Sunny | Cool | Normal | Weak | Yes |
| Sunny | Mild | Normal | Strong | Yes |

**OUTLOOK = "SUNNY"**

| Outlook | Temperature | Humidity | Windy | PlayTennis |
|---|---|---|---|---|
| Rainy | Mild | High | Weak | Yes |
| Rainy | Cool | Normal | Weak | Yes |
| Rainy | Cool | Normal | Strong | No |
| Rainy | Mild | Normal | Weak | Yes |
| Rainy | Mild | High | Strong | No |

**OUTLOOK = "RAINY"**

| Outlook | Temperature | Humidity | Windy | PlayTennis |
|---|---|---|---|---|
| Sunny | Hot | High | Weak | No |
| Sunny | Hot | High | Strong | No |
| Sunny | Mild | High | Weak | No |
| Sunny | Cool | Normal | Weak | Yes |
| Sunny | Mild | Normal | Strong | Yes |

P= 2 N= 3 Total= 5

- ENTROPY:

$$Entropy = \frac{-p}{p+n} log_2(\frac{p}{p+n}) - \frac{n}{p+n} log_2(\frac{n}{p+n})$$

$$Entropy(S_{sunny}) = \frac{-2}{2+3} log_2(\frac{2}{2+3}) - \frac{3}{2+3} log_2(\frac{3}{2+3})$$

**=>0.971**

- For each Attribute: (let say **Humidity**):
  - Calculate Entropy for each Humidity, i.e for 'High' and 'Normal'

| Outlook | Humidity | PlayTennis |
|---|---|---|
| Sunny | High | No |
| Sunny | High | No |
| Sunny | High | No |
| Sunny | Normal | Yes |
| Sunny | Normal | Yes |

| Humidity | p | n | Entropy |
|---|---|---|---|
| high | 0 | 3 | 0 |
| normal | 2 | 0 | 0 |

  - Calculate **Average Information Entropy**: I(Humidity) = 0
  - Calculate **Gain:** Gain = 0.971

- For each Attribute: (let say **Windy**):
  - Calculate Entropy for each Windy, i.e for 'Strong' and 'Weak'

| Outlook | Windy | PlayTennis |
|---|---|---|
| Sunny | Strong | No |
| Sunny | Strong | Yes |
| Sunny | Weak | No |
| Sunny | Weak | No |
| Sunny | Weak | Yes |

| Windy | p | n | Entropy |
|---|---|---|---|
| Strong | 1 | 1 | 1 |
| Weak | 1 | 2 | 0.918 |

  - Calculate **Average Information Entropy**: I(Windy) = 0.951
  - Calculate **Gain:** Gain = 0.020

- For each Attribute: (let say **Temperature**):
  - Calculate Entropy for each Windy, i.e for 'Cool', 'Hot' and 'Mild'

| Outlook | Temperature | PlayTennis |
|---|---|---|
| Sunny | Cool | Yes |
| Sunny | Hot | No |
| Sunny | Hot | No |
| Sunny | Mild | No |
| Sunny | Mild | Yes |

| Temperature | p | n | Entropy |
|---|---|---|---|
| Cool | 1 | 0 | 0 |
| Hot | 0 | 2 | 0 |
| Mild | 1 | 1 | 1 |

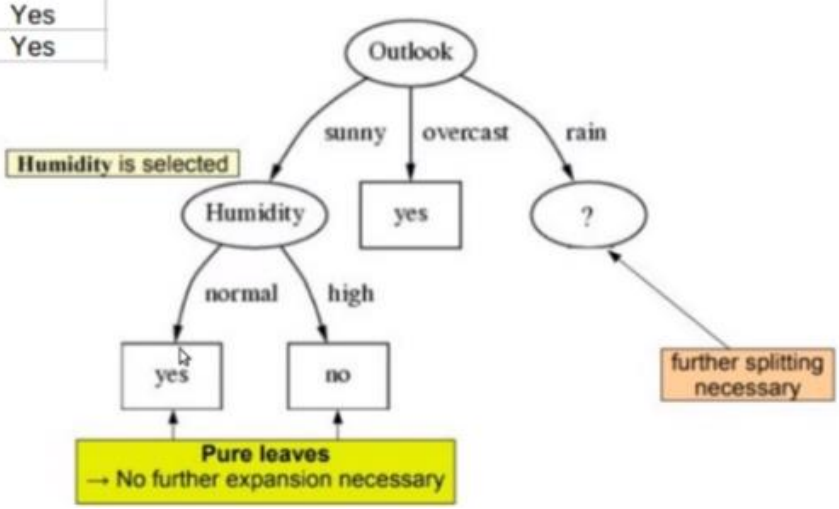
  - Calculate **Average Information Entropy**: I(Temp) = 0.4
  - Calculate **Gain:** Gain = 0.571

- PICK THE HIGHEST GAIN ATTRIBUTE.

| Attributes | Gain |
|---|---|
| Temperature | 0.571 |
| Humidity | 0.971 |
| Windy | 0.02 |

## NEXT NODE IN SUNNY:
## HUMIDITY

| Outlook | Humidity | PlayTennis |
|---|---|---|
| Sunny | High | No |
| Sunny | High | No |
| Sunny | High | No |
| Sunny | Normal | Yes |
| Sunny | Normal | Yes |




| Outlook | Temperature | Humidity | Windy | PlayTennis |
|---|---|---|---|---|
| Rainy | Mild | High | Weak | Yes |
| Rainy | Cool | Normal | Weak | Yes |
| Rainy | Cool | Normal | Strong | No |
| Rainy | Mild | Normal | Weak | Yes |
| Rainy | Mild | High | Strong | No |

P= 3 N= 2

Total = 5

- ENTROPY:

$$Entropy = \frac{-p}{p+n} log_2(\frac{p}{p+n}) - \frac{n}{p+n} log_2(\frac{n}{p+n})$$

$$Entropy(S_{Rainy}) = \frac{-3}{3+2} log_2(\frac{3}{3+2}) - \frac{2}{3+2} log_2(\frac{2}{2+3})$$

**=>0.971**

- For each Attribute: (let say **Humidity**):
  - Calculate Entropy for each Humidity, i.e for 'High' and 'Normal'

| Outlook | Humidity | PlayTennis |
|---|---|---|
| Rainy | High | Yes |
| Rainy | High | No |
| Rainy | Normal | Yes |
| Rainy | Normal | No |
| Rainy | Normal | Yes |

| Attribute | p | n | Entropy |
|---|---|---|---|
| High | 1 | 1 | 1 |
| Normal | 2 | 1 | 0.918 |

  - Calculate **Average Information Entropy**: I(Humidity) = 0.951
  - Calculate **Gain:** Gain = 0.020

- For each Attribute: (let say **Windy**):
  - Calculate Entropy for each Windy, i.e for 'Strong' and 'Weak'

| Outlook | Windy | PlayTennis |
|---|---|---|
| Rainy | Strong | No |
| Rainy | Strong | No |
| Rainy | Weak | Yes |
| Rainy | Weak | Yes |
| Rainy | Weak | Yes |

| Attribute | p | n | Entropy |
|---|---|---|---|
| Strong | 0 | 2 | 0 |
| Weak | 3 | 0 | 0 |

  - Calculate **Average Information Entropy**: I(Windy) = 0
  - Calculate **Gain:** Gain = 0.971

- For each Attribute: (let say **Temperature**):
  - Calculate Entropy for each Windy, i.e for 'Cool', 'Hot' and 'Mild'

| Outlook | Temperature | PlayTennis |
|---|---|---|
| Rainy | Mild | Yes |
| Rainy | Cool | Yes |
| Rainy | Cool | No |
| Rainy | Mild | Yes |
| Rainy | Mild | No |

| Attribute | p | n | Entropy |
|---|---|---|---|
| Cool | 1 | 1 | 1 |
| Mild | 2 | 1 | 0.918 |

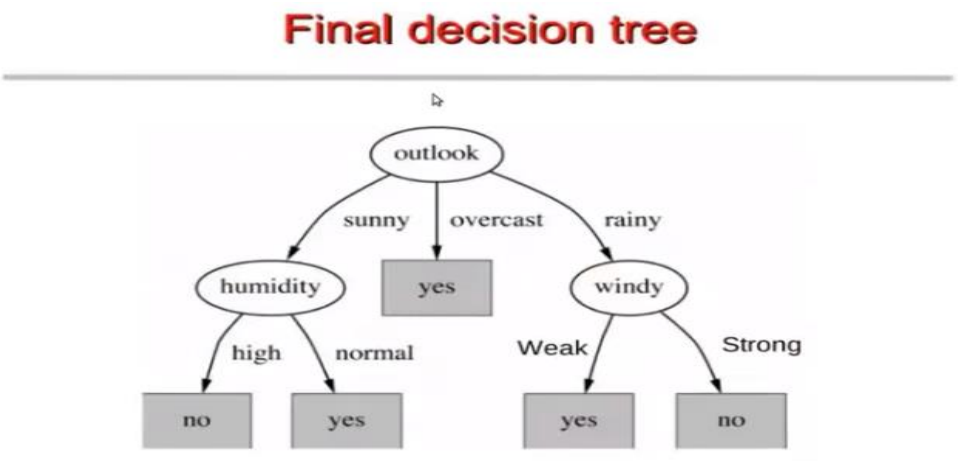
  - Calculate **Average Information Entropy**: I(Temp) = 0.951
  - Calculate **Gain:** Gain = 0.020

- PICK THE HIGHEST GAIN ATTRIBUTE.

| Attributes | Gain |
|---|---|
| Humidity | 0.02 |
| Windy | 0.971 |
| Temperature | 0.02 |

NEXT NODE IN RAINY:

**WINDY**




## Advantages of the Decision Tree

- It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
- It can be very useful for solving decision-related problems.
- It helps to think about all the possible outcomes for a problem.
- There is less requirement of data cleaning compared to other algorithms.

## Disadvantages of the Decision Tree

- The decision tree contains lots of layers, which makes it complex.
- It may have an overfitting issue, which can be resolved using the **Random Forest algorithm.**
- For more class labels, the computational complexity of the decision tree may increase

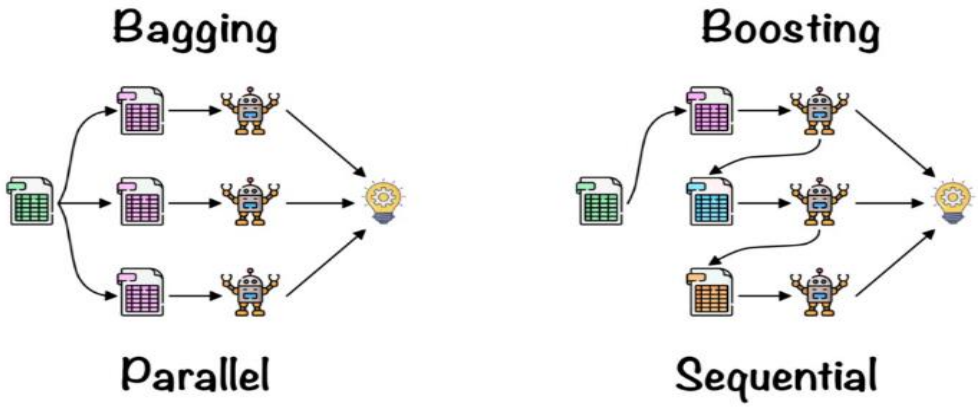
## Issues In Decision Tree Learning

1. Overfitting the Data
2. Incorporating Continuous valued attributes
3. Handling training examples with missing attribute values
4. Handling attributes with different costs
5. Alternative measures for selecting attributes

# Ensemble Method: Introduction to Random Forest, Accuracy measure & performance

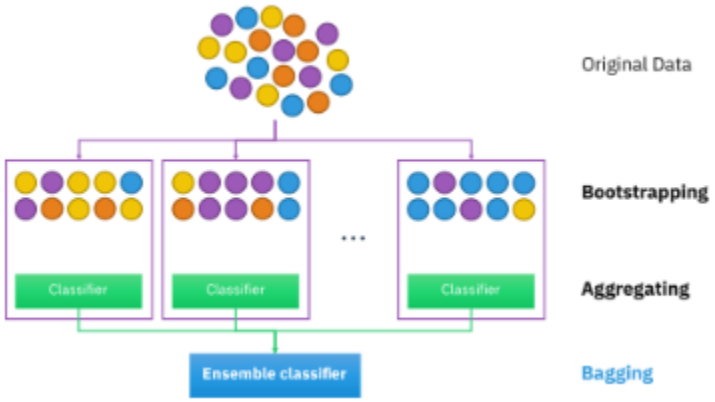
## Random Forest Algorithm:

- Random forest is a ***Supervised Machine Learning Algorithm*** that is ***used widely in Classification and Regression problems***. It builds decision trees on different samples and takes their majority vote for classification and average in case of regression.
- One of the most important features of the Random Forest Algorithm is that it can handle the data set containing ***continuous variables*** as in the case of regression and ***categorical variables*** as in the case of classification. It performs better results for classification problems.
- Random forest works on ensemble technique. ***Ensemble*** simply means combining multiple models. Thus a collection of models is used to make predictions rather than an individual model.
- *Ensemble uses two types of methods*:
- 1. **Bagging**– It creates a different training subset from sample training data with replacement & the final output is based on majority voting. For example, Random Forest.
- 2. **Boosting**– It combines weak learners into strong learners by creating sequential models such that the final model has the highest accuracy. For example, ADA BOOST, XG BOOST




- Bagging, also known as ***Bootstrap Aggregation*** is the ensemble technique used by random forest. Bagging chooses a random sample from the data set. Hence each model is generated from the samples (Bootstrap Samples) provided by the Original Data with replacement known as ***row sampling***. This step of row sampling with replacement is

called ***bootstrap***. Now each model is trained independently which generates results. The final output is based on majority voting after combining the results of all models. This step which involves combining all the results and generating output based on majority voting is known as ***aggregation***.



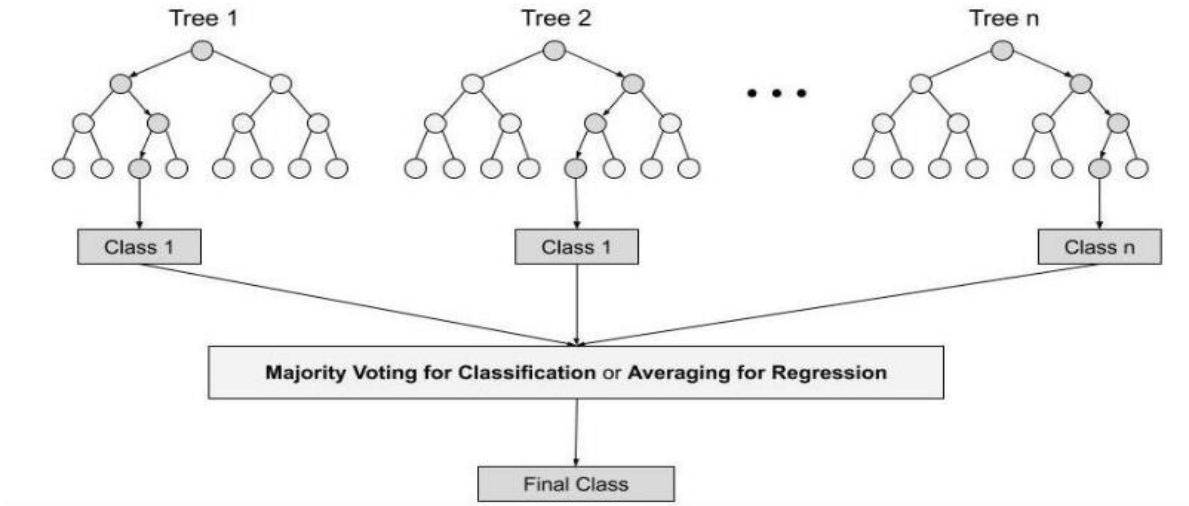

## Steps involved in random forest algorithm:

Step 1: In Random forest n number of random records are taken from the data set having k number of records.

Step 2: Individual decision trees are constructed for each sample.

Step 3: Each decision tree will generate an output.

Step 4: Final output is considered based on ***Majority Voting or Averaging*** for Classification and regression respectively.



## Important Features of Random Forest

**1. Diversity-** Not all attributes/variables/features are considered while making an individual tree, each tree is different.

**2. Immune to the curse of dimensionality-** Since each tree does not consider all the features, the feature space is reduced.

**3. Parallelization-**Each tree is created independently out of different data and attributes. This means that we can make full use of the CPU to build random forests.

**4. Train-Test split-** In a random forest we don't have to segregate the data for train and test as there will always be 30% of the data which is not seen by the decision tree.

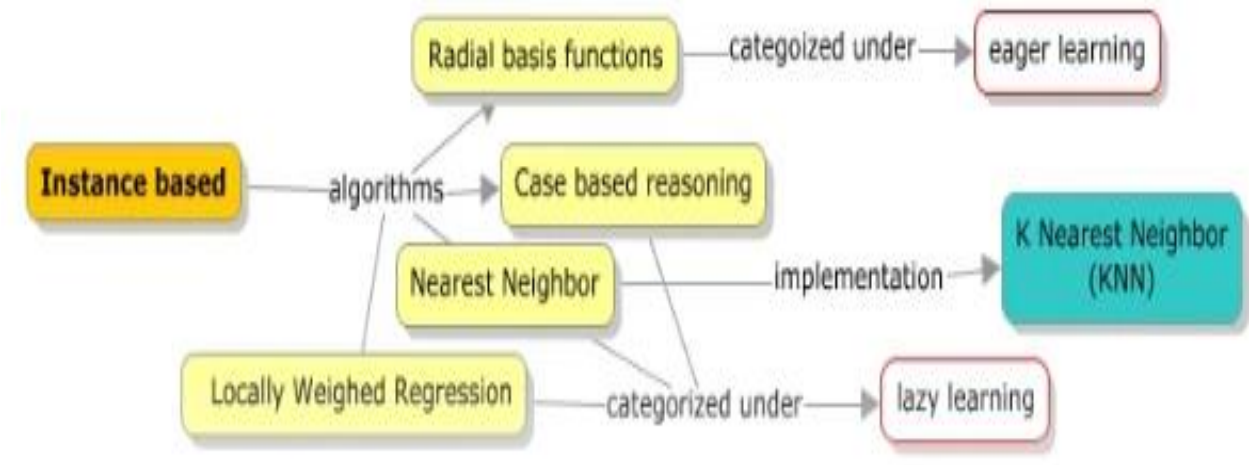
**5. Stability-** Stability arises because the result is based on majority voting/ averaging

| Decision trees | Random Forest |
|---|---|
| 1. Decision trees normally suffer from the problem of overfitting if it's allowed to grow without any control. | 1. Random forests are created from subsets of data and the final output is based on average or majority ranking and hence the problem of overfitting is taken care of. |
| 2. A single decision tree is faster in computation. | 2. It is comparatively slower. |
| 3. When a data set with features is taken as input by a decision tree it will formulate some set of rules to do prediction. | 3. Random forest randomly selects observations, builds a decision tree and the average result is taken. It doesn't use any set of formulas. |

# 5.3 Instance based learning- Introduction, KNN algorithm, Distance measures, model building, locally weighted regression, radial basis functions, SVM classifier, hyper-plane, slack variables, geometric transformation kernel trick, kernel transformation.

## 5.3.1 Introduction

- ***Generalization* —** usually refers to a **ML** model's ability to perform well on new unseen data rather than just the data that it was trained on.
- Most Machine Learning tasks are about making predictions. This means that given a number of training examples, the system needs to be able to make good **"predictions for" / "generalize to"** examples it has never seen before.
- There are two main approaches to generalization: ***instance-based learning*** and ***model-based learning***
- Instance-based learning algorithms, as the name suggests, learn from examples. That is, the algorithm looks at a set of training data and tries to find a pattern that can be generalized to new data. This pattern is then used to make predictions on new data.
- Model-based learning algorithms, on the other hand, learn from a model that is created from the training data. This model can be thought of as a mathematical representation of the training data. The model is then used to make predictions on new data.

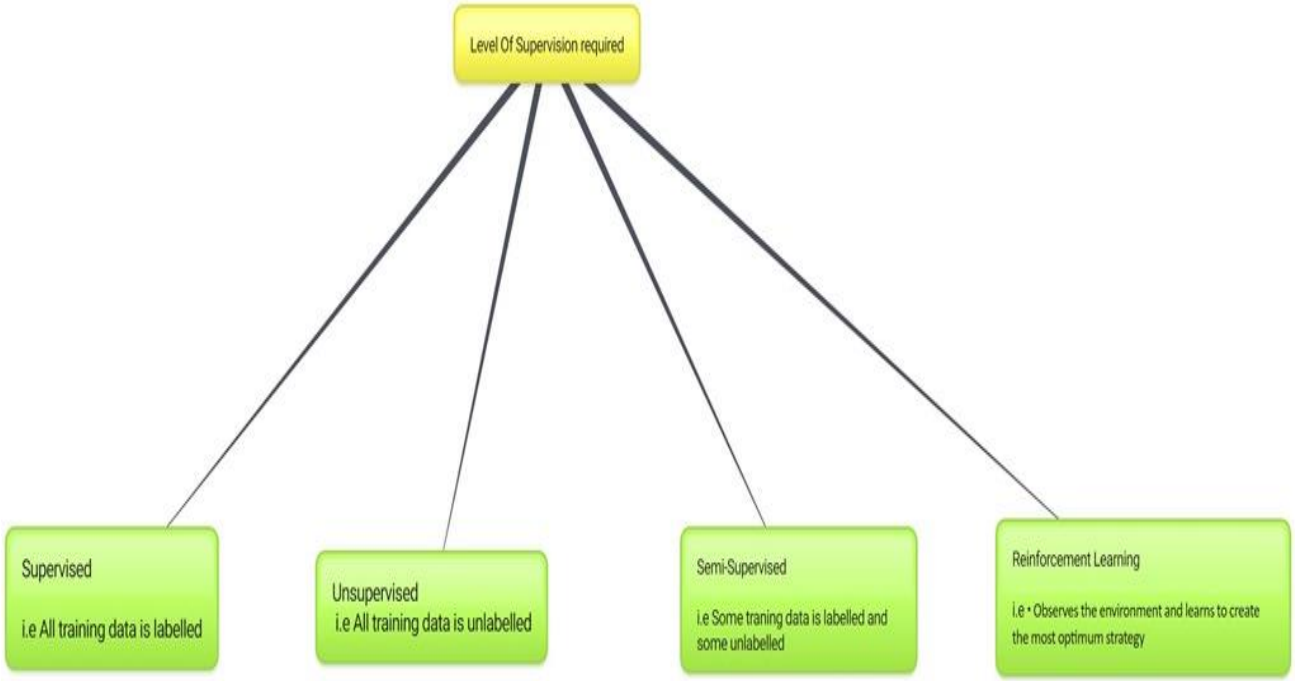


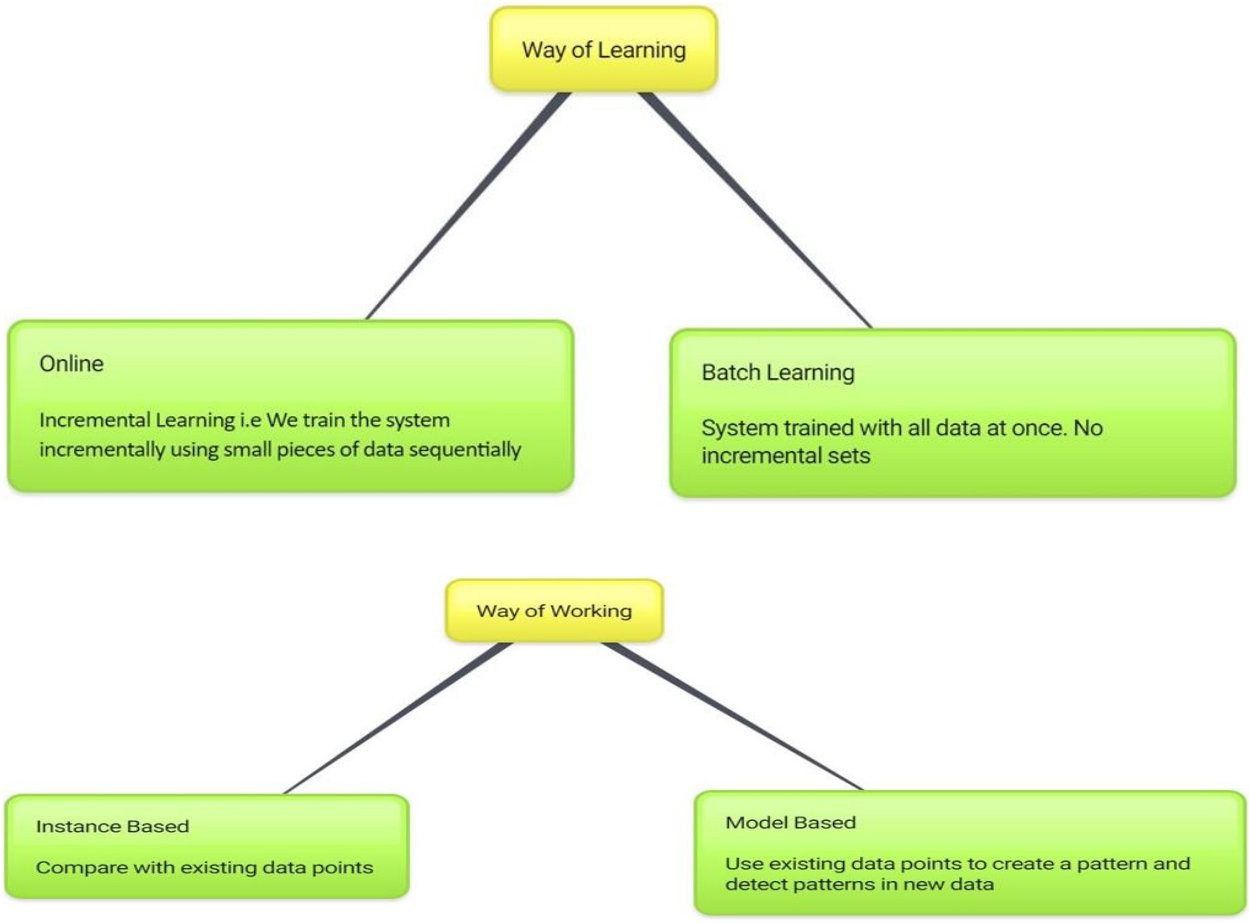
# Machine Learning Algorithms

- Supervised
- Unsupervised
- Semi-Supervised
- Reinforcement Learning

- Instance-based Learning
- Model-based Learning

- Online Learning
- Batch Learning





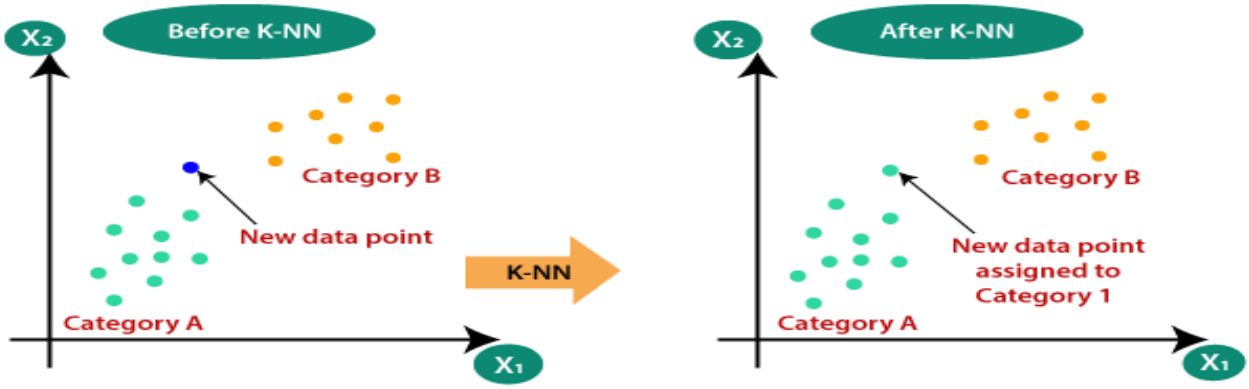

## 5.3.2 KNN algorithm

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- 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.
- **K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.**
- 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.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.
- **Lazy learning algorithm** − KNN is a lazy learning algorithm because it does not have a specialized training phase and uses all the data for training while classification.

- **Non-parametric learning algorithm** − KNN is also a non-parametric learning algorithm because it doesn't assume anything about the underlying data.

## Example

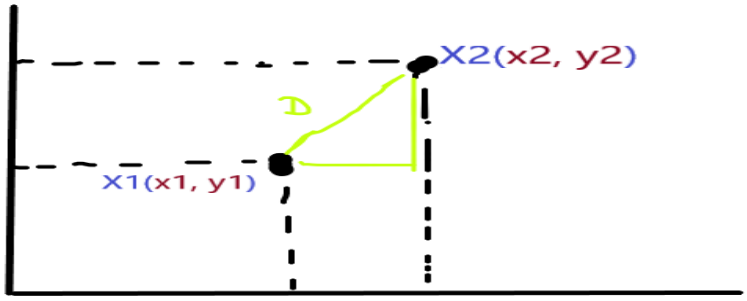
- Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.


**Euclidean distance**

We mostly use this distance measurement technique to find the distance between consecutive points. It is generally used to find the distance between two real-valued vectors. Euclidean distance is used when we have to calculate the distance of real values like integer, float, etc… One issue with this distance measurement technique is that we have to normalize or standard the data into a balance scale otherwise the outcome will not preferable.

Let's take an example of a 2-Dimensional vector and take a geometrical intuition of distance measures on 2-Dim data for a better understanding.




From the above image, you can see that there are 2-Dim data $X_1$ and $X_2$ placed at certain coordinates in 2 dimensions, suppose $X_1$ is at $(x_1,y_1)$ coordinates and $X_2$ is at $(x_2,y_2)$ coordinates. We have 2-dim data so we considered F1 and F2 two features and D is considered as the shortest line from $X_1$ and $X_2$, If we find the distance between these data points that can be found by the Pythagoras theorem and can be written as:

$$D = \sqrt{[(x2 - x1)^2 + (y2 - y1)^2]}$$

You already know that we are finding the distance or we can say the length of any 2 points, so we in vectorize from we can write that as $d = \| X1 - X2 \|$

Suppose we take D-dimensional vector it means $X_1$ belongs to $R^d$ ($X_1 \in R^d$) and also $X_2$ belongs to $R^d$ ($X_2 \in R^d$) so the distance between X1 and X2 is written as:

$$d = \sqrt{\sum_{i=1}^{N} (Xi - Yi)^2}$$

**Example**: K nearest neighbor classifier to predict the diabetic patient with the given features BMI, Age. If the training examples are

| BMI | Age | Sugar |
|---|---|---|
| **33.6** | **50** | **1** |
| **26.6** | **30** | **0** |
| **23.4** | **40** | **0** |
| **43.1** | **67** | **0** |
| **35.3** | **23** | **1** |
| **35.9** | **67** | **1** |
| **36.7** | **45** | **1** |
| **25.7** | **46** | **0** |
| **23.3** | **29** | **0** |
| **31** | **56** | **1** |

**0**-Non-diabetic

**1**-Diabetic

**Test Example BMI=43.6, Age=40, Sugar=?**

The given training dataset has 10 instances with two features BMI (Body Mass Index) and Age. Sugar is the target label. The target label has two possibilities 0 and 1. 0 means the diabetic patient has no sugar and 1 means the diabetic patient has sugar.

Given the dataset and new test instance, we need to find the distance from the new test instance to every training example. Here we use the euclidean distance formula to find the distance.

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

If we choose K=3 then we need to select three neighbours which are close to the given test data

| BMI | Age | Sugar | Formula | Distance |
|---|---|---|---|---|
| 33.6 | 50 | 1 | √((43.6-33.6)^2+(40-50)^2 ) | 14.14 |
| 26.6 | 30 | 0 | √((43.6-26.6)^2+(40-30)^2 ) | 19.72 |
| 23.4 | 40 | 0 | √((43.6-23.4)^2+(40-40)^2 ) | 20.20 |
| 43.1 | 67 | 0 | √((43.6-43.1)^2+(40-67)^2 ) | 27.00 |
| 35.3 | 23 | 1 | √((43.6-35.3)^2+(40-23)^2 ) | 18.92 |
| 35.9 | 67 | 1 | √((43.6-35.9)^2+(40-67)^2 ) | 28.08 |
| 36.7 | 45 | 1 | √((43.6-36.7)^2+(40-45)^2 ) | 8.52 |
| 25.7 | 46 | 0 | √((43.6-25.7)^2+(40-46)^2 ) | 18.88 |
| 23.3 | 29 | 0 | √((43.6-23.3)^2+(40-29)^2 ) | 23.09 |
| 31 | 56 | 1 | √((43.6-31)^2+(40-56)^2 ) | 20.37 |

point i.e**,(43.6,40**).

Select three data points based on minimum Euclidean distance

| BMI | Age | Sugar |
|---|---|---|
| 36.7 | 45 | 1 |
| 33.6 | 50 | 1 |
| 25.7 | 46 | 0 |

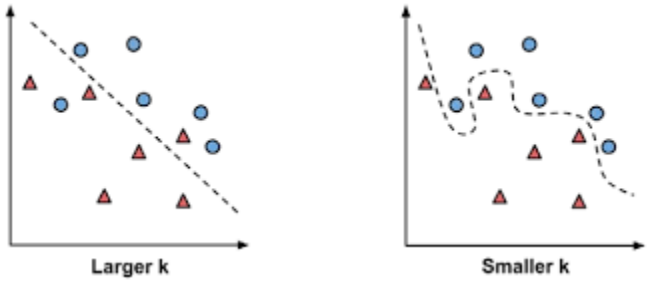
Now, we need to apply the majority voting technique to decide the resulting label for the test data point**,(43.6,40**).

As majority is classified as **1** then the **test data point** is classified as **1**(i.e, patient has sugar)

Test Example BMI=43.6, Age=40, Sugar=1

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

- There is no particular way to determine the best value for “K”, Choosing a right value of K is a process called Hyperparameter Tuning**.**
- 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.
- The impact of selecting a smaller or larger K value on the model
- **Larger K value:** The case of underfitting occurs when the value of k is increased. In this case, the model would be unable to correctly learn on the training data.
- **Smaller k value:** The condition of overfitting occurs when the value of k is smaller. The model will capture all of the training data, including noise. The model will perform poorly for the test data in this scenario.




**Few ways of selecting K value**

1. **Square Root Method**: Take square root of the number of samples in the training dataset.
2. **Cross Validation Method:** We should also use cross validation to find out the optimal value of K in KNN. Start with K=1, run cross validation (5 to 10 fold), measure the accuracy and keep repeating till the results become consistent.
   K=1, 2, 3... As K increases, the error usually goes down, then stabilizes, and then raises again. Pick the optimum K at the beginning of the stable zone. This is also called **Elbow Method.**
3. **Domain Knowledge** also plays a vital role while choosing the optimum value of K.
4. K should be an **odd number**

## Advantages of KNN Algorithm

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

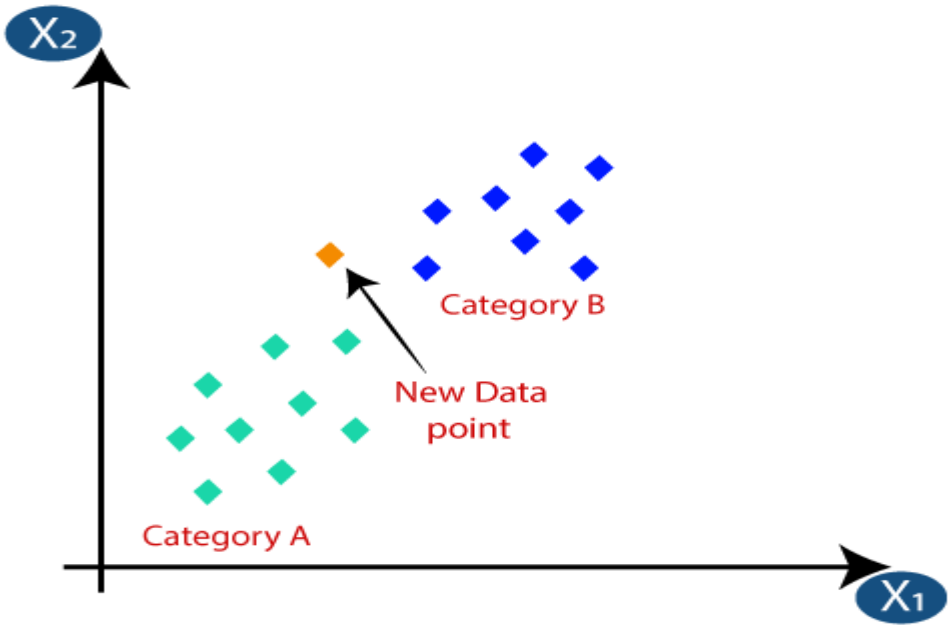
## Disadvantages of KNN Algorithm

- Always needs to determine the value of K which may be complex some time.

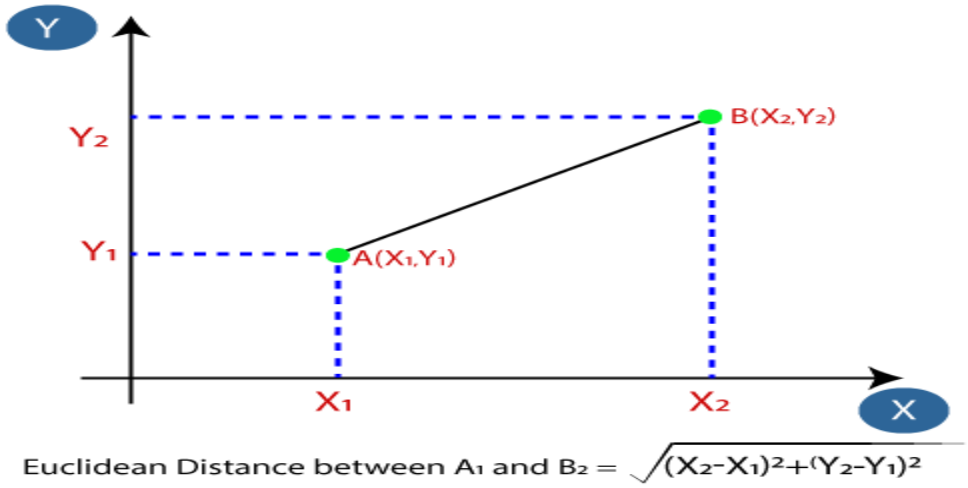
- The computation cost is high because of calculating the distance between the data points for all the training samples.

## The K-NN working can be explained on the basis of the below algorithm:

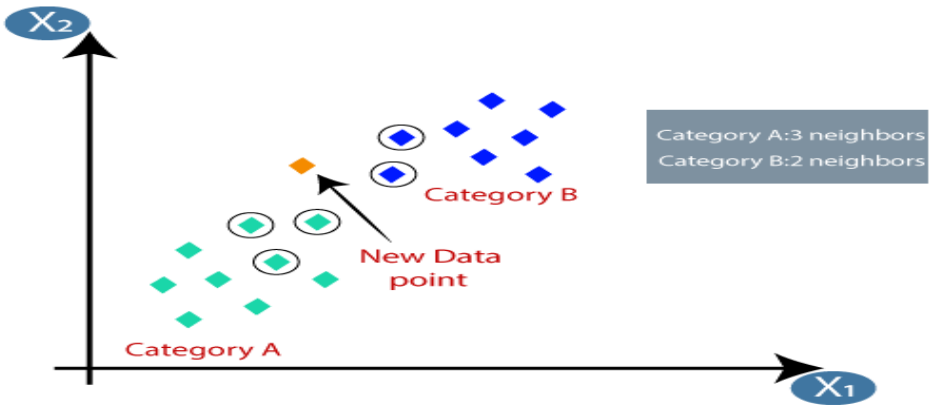
- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.




- Firstly, we will choose the number of neighbors, so we will choose the k=5.
- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



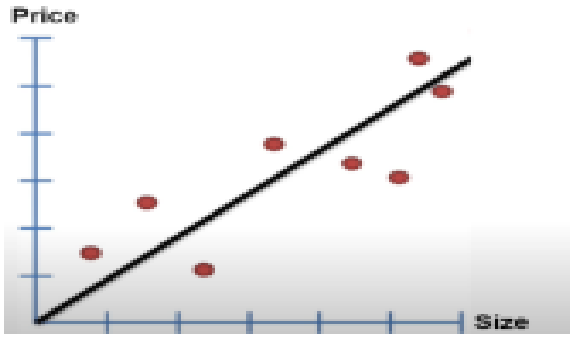

- By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:



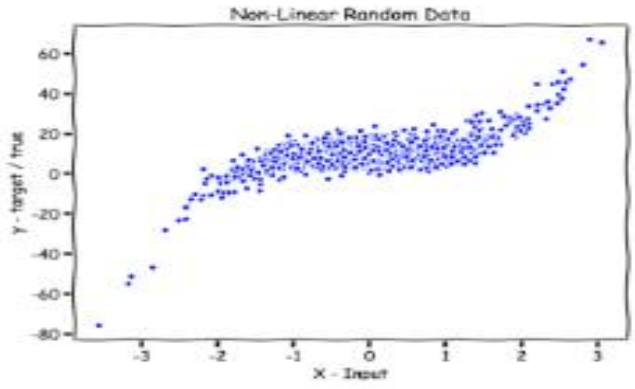

- As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.
- Below are some points to remember while selecting the value of K in the K-NN algorithm:
- There is no particular way to determine the best value for "K", so we need to try some values to 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.

## 5.3.3 locally weighted regression

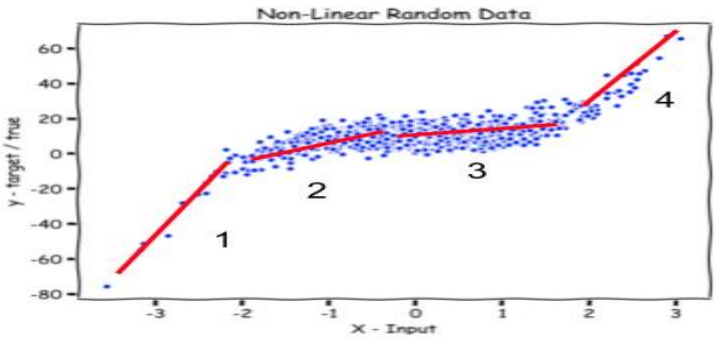
- It is a non-parametric algorithm, unlike a typical linear regression algorithm which is a parametric algorithm. A parametric algorithm is an algorithm that doesn't need to retain the training data when we need to make predictions.
- In the traditional linear regression algorithm, we aim to fit a general line on the given training dataset and predict some parameters that can be generalized for any input to make correct predictions.




- So this is what we do in a linear regression algorithm. Then the question arises why do we need another regression algorithm like a locally weighted regression algorithm?
- A standard line wouldn't fit entirely into this type of dataset. In such a case, we have to introduce a locally weighted algorithm that can predict a very close value to the actual value of a given query point.




- So another thought coming to our mind is, let's break the given dataset into a smaller dataset and say we have multiple smaller lines that can fit the smaller dataset. Together, these multiple lines fit the entire dataset. To understand better, look at the diagram below.




- Look at the 4 small lines that have been tried to fit smaller datasets and fit the entire dataset together. But now, the question arises of how to select the best line that can be chosen to predict the output for a given query point

## 5.3.4 Radial basis functions

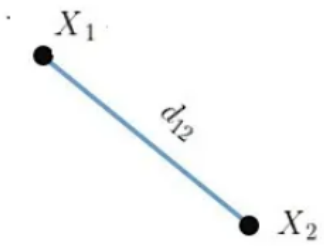
• RBF Kernel is popular because of its similarity to K-Nearest Neighborhood Algorithm. It has the advantages of K-NN and overcomes the space complexity problem as RBF Kernel Support Vector Machines just needs to store the support vectors during training and not the entire dataset.
• RBF kernels are the most generalized form of kernelization and is one of the most widely used kernels due to its similarity to the Gaussian distribution. The RBF kernel function for two points $X_1$ and $X_2$ computes the similarity or how close they are to each other. This kernel can be mathematically represented as follows:

$$K(X_1, X_2) = exp(-\frac{||X_1 - X_2||^2}{2\sigma^2})$$

where,

1. 'σ' is the variance and our hyperparameter
2. $||X_1 - X_2||$ is the Euclidean ($L_2$-norm) Distance between two points $X_1$ and $X_2$

Let $d_{12}$ be the distance between the two points $X_1$ and $X_2$, we can now represent $d_{12}$ as follows:



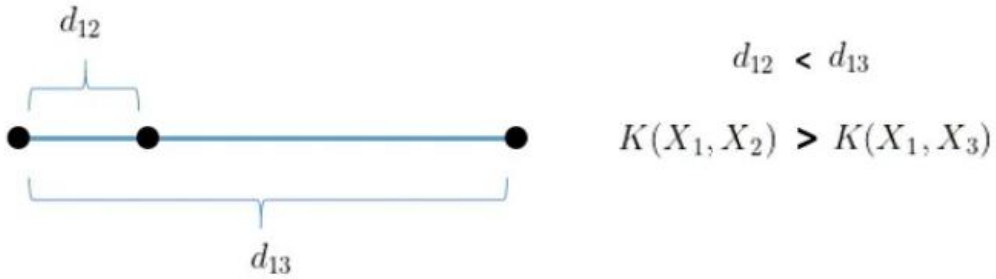

$$d_{12} = ||X_1 - X_2||^2$$

Distance between two points in space

The kernel equation can be re-written as follows:

$$K(X_1, X_2) = exp(-\frac{d_{12}}{2\sigma^2})$$

The maximum value that the RBF kernel can be is 1 and occurs when $d_{12}$ is 0 which is when the points are the same, i.e. $X_1 = X_2$.
When the points are the same, there is no distance between them and therefore they are extremely similar.
When the points are separated by a large distance, then the kernel value is less than 1 and close to 0 which would mean that the points are dissimilar.
Distance can be thought of as an equivalent to dissimilarity because we can notice that when distance between the points increases, they are less similar.



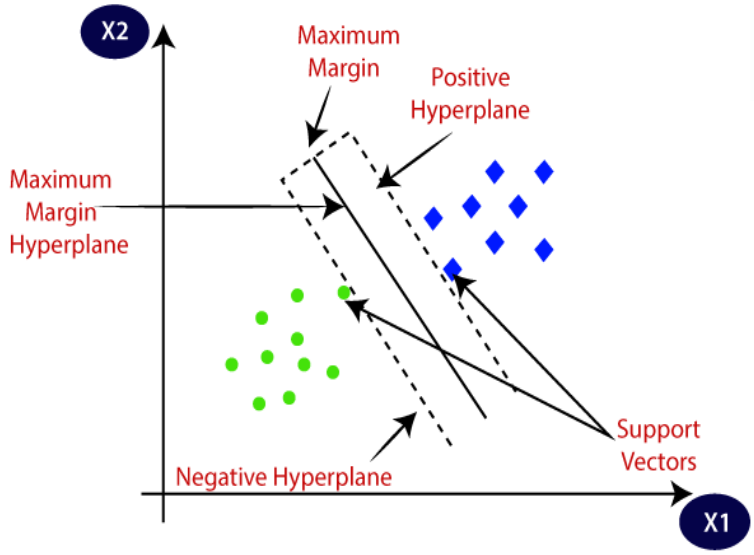

Similarity decreases as distance increases

- This particular type of function is useful in cases where data may need to be classified in a non-linear way.

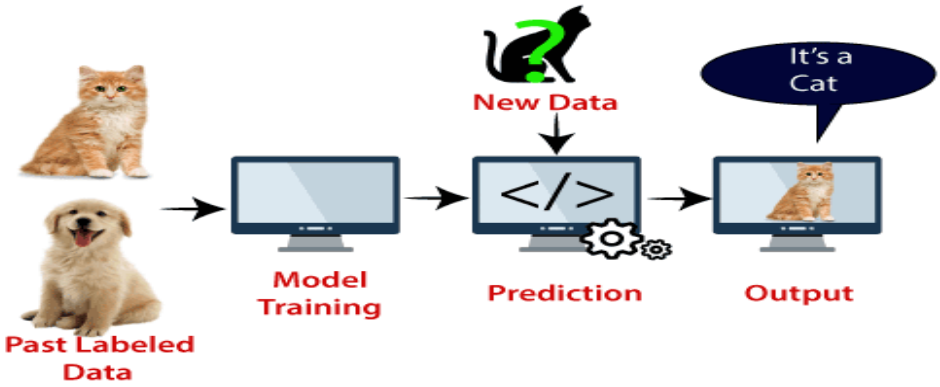
## 5.3.5 Support Vector Machine

- Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.
- The goal of the SVM algorithm is to create **the best line or decision boundary** that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a **hyperplane.**


- Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.
- The goal of the SVM algorithm is to create **the best line or decision boundary** that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a **hyperplane.**




- **Example:** SVM can be understood with the example that we have used in the KNN classifier. Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm. We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature. So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case of cat and dog. On the basis of the support vectors, it will classify it as a cat. Consider the below diagram:




- ## SVM can be of two types:

- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.

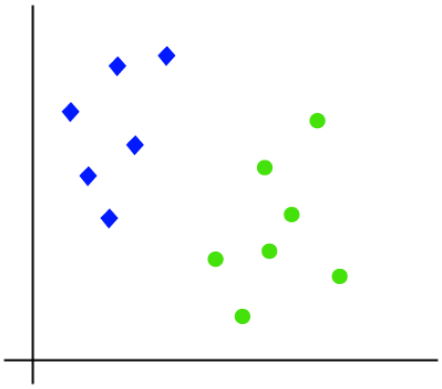
- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.
- **Hyperplane:** There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.
- The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.
- We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

## Support Vectors:

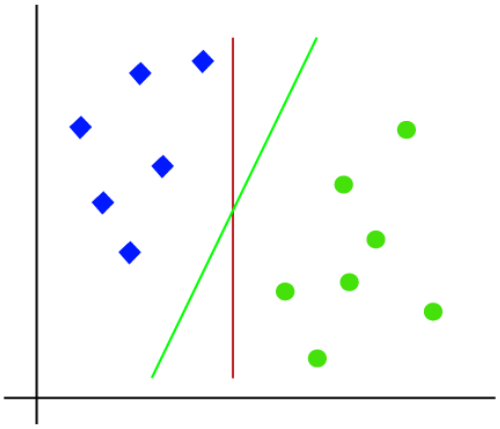
The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

## Linear SVM:

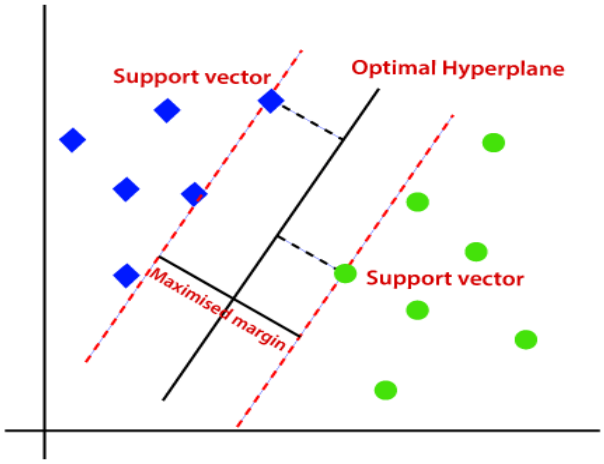
The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image:



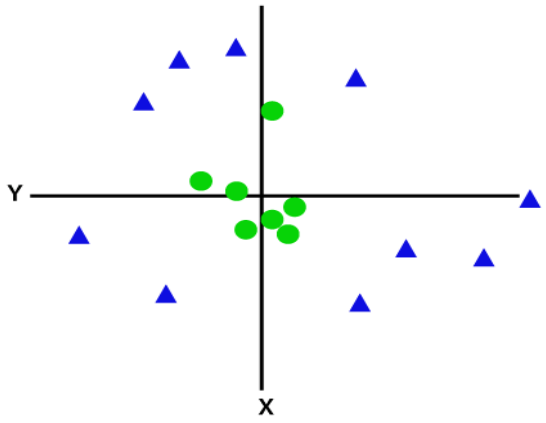
- So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image:



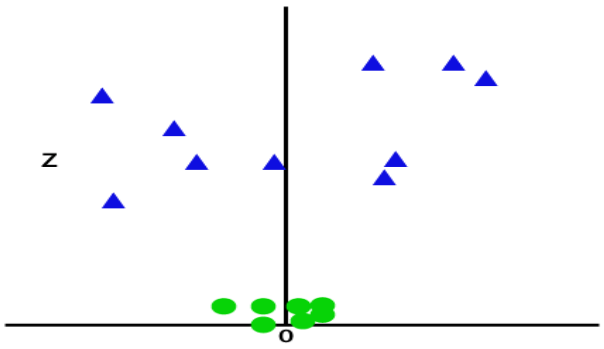
- Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as **margin**. And the goal of SVM is to maximize this margin. The **hyperplane** with maximum margin is called the **optimal hyperplane**.



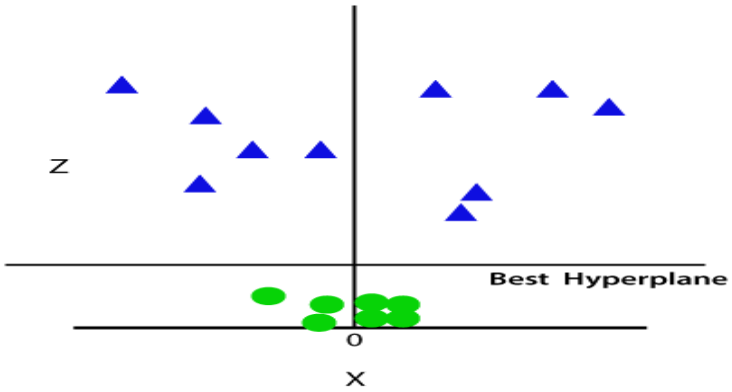

- **Non-Linear SVM:**
- If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:



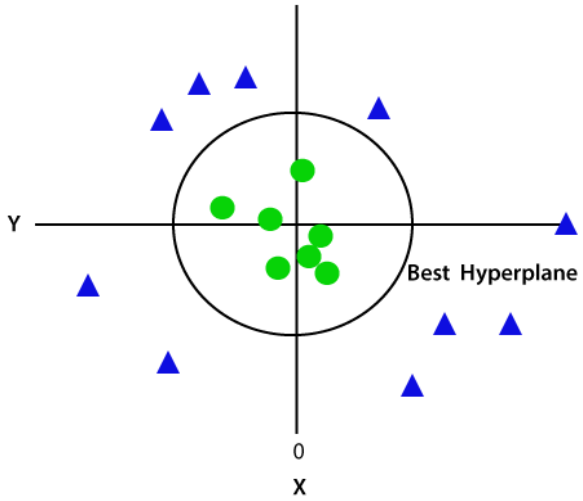

- So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:
- $z=x^2 +y^2$




So now, SVM will divide the datasets into classes in the following way. Consider the below image:




Since we are in 3-d Space, hence it is looking like a plane parallel to the x-axis. If we convert it in 2d space with z=1, then it will become as:

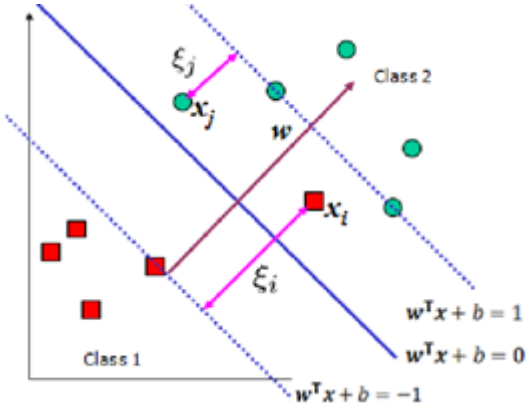



## Concept of Slack Variable:

The concept of slack variables is used in the formulation of the "soft-margin" SVM to handle cases where data is not linearly separable, or when one allows for some degree of error in classification.

Mathematically, for each data point i, a slack variable $\xi i \geq 0$ is introduced. The slack variable $\xi i$ measures the degree of misclassification of the data point xi.

- **$\xi i = 0$ if xi is on the correct side of the margin.**
- **$0 < \xi i < 1$ if xi is on the correct side of the hyperplane but on the wrong side of the margin.**
- **$\xi i \geq 1$ if xi is on the wrong side of the hyperplane, i.e., it is misclassified.**

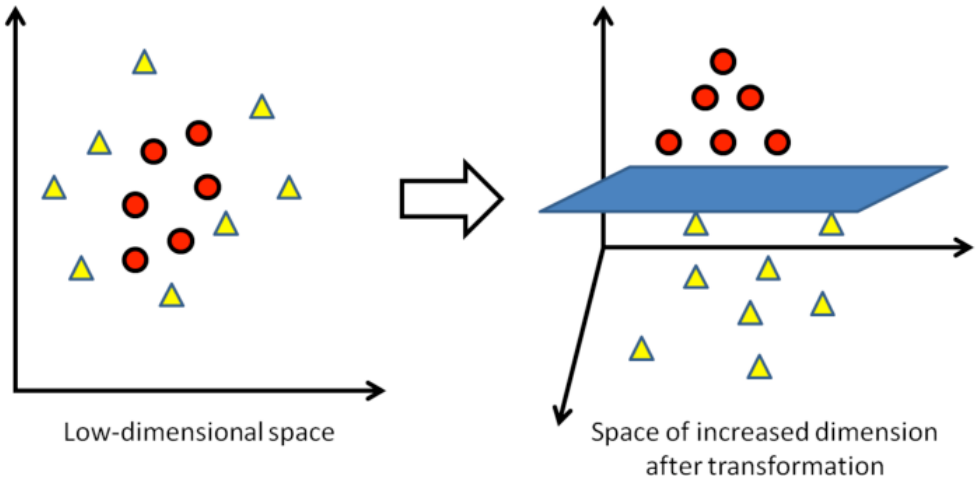



## What is Kernel Trick ?

The "Kernel Trick" is a method used in Support Vector Machines (SVMs) to convert data (that is not linearly separable) into a higher-dimensional feature space where it may be linearly separated.

This technique enables the SVM to identify a hyperplane that separates the data with the maximum margin, even when the data is not linearly separable in its original space. The kernel functions are used to compute the inner product between pairs of points in the transformed feature space without explicitly computing the transformation itself. This makes it computationally efficient to deal with high dimensional feature spaces.

The most widely used kernels in SVM are the linear kernel, polynomial kernel, and Gaussian (radial basis function) kernel.

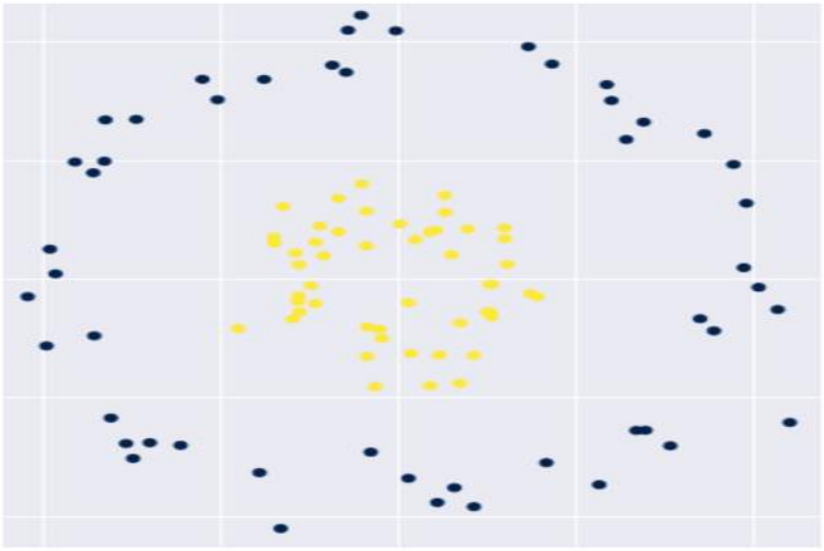



## 5.3.6 kernel transformation

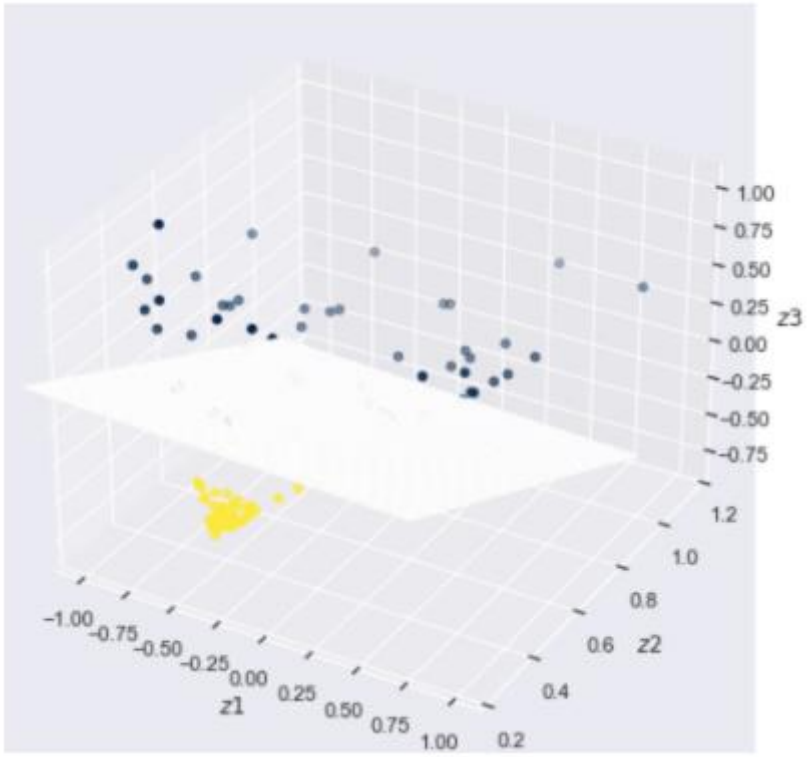
**In machine learning, a kernel refers to a method that allows us to apply linear classifiers to non-linear problems by mapping non-linear data into a higher-dimensional space without the need to visit or understand that higher-dimensional space.**

**Why Do We Need a Kernel?**

Suppose you have a 2-dimensional dataset that contains 2 classes of observations, and you need to find a function that separates the two classes. The data is not linearly separable in a 2-dimensional space



We would have to fit a polynomial function to separate the data, which complicates our classification problem. But what if we could transform this data into a higher-dimensional (3D) space where the data is separable by a linear classifier?




If we can find a mapping from the 2D space into the 3D space where our observations are linearly separable, we can

1. Transform our data from 2D into 3D
2. Find a linear decision boundary by fitting a linear classifier (a plane separating the data) in the 3D space
3. Map the linear decision boundary back into 2D space. The result will be a non-linear decision boundary in 2D

So we've essentially found a non-linear decision boundary while only doing the work of finding a linear classifier.

To build a linear classifier, we've had to transform our data into 3D space. Doesn't this make the whole operation significantly more complicated?

**What is a Kernel?**

We've established that a Kernel helps us separate data with a non-linear decision boundary using a linear classifier. To understand how this works, let's briefly recall how a linear classifier works.

**Role of the Dot Product in Linear Classifiers**

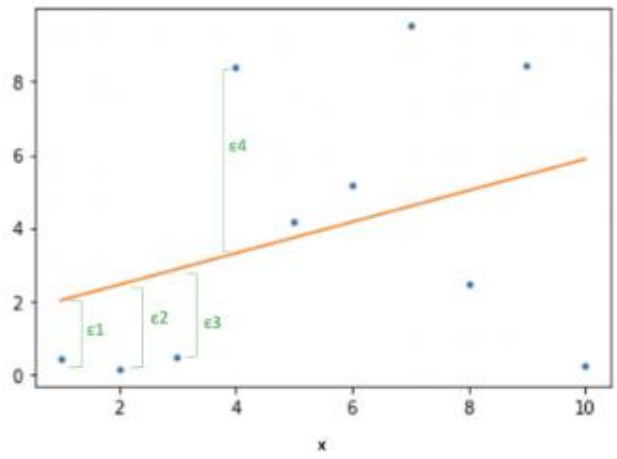
Consider a linear regression model of the following form:

$$y_i = w_0 + w_1 x_i + w_2 x_i + \epsilon_1$$

If we package all weights into a vector w = {w_0, w_1, w_2} we can express this as a simple dot product between the weight vector and the observation x_i.

$$y_i = w^T x_i + \epsilon_1$$

The dot product between x_I and the weights gives us the predicted point on the line for the actual observation y_i. The difference is the error ε_1




In other words, the dot product is central to the prediction operation in a linear classifier.

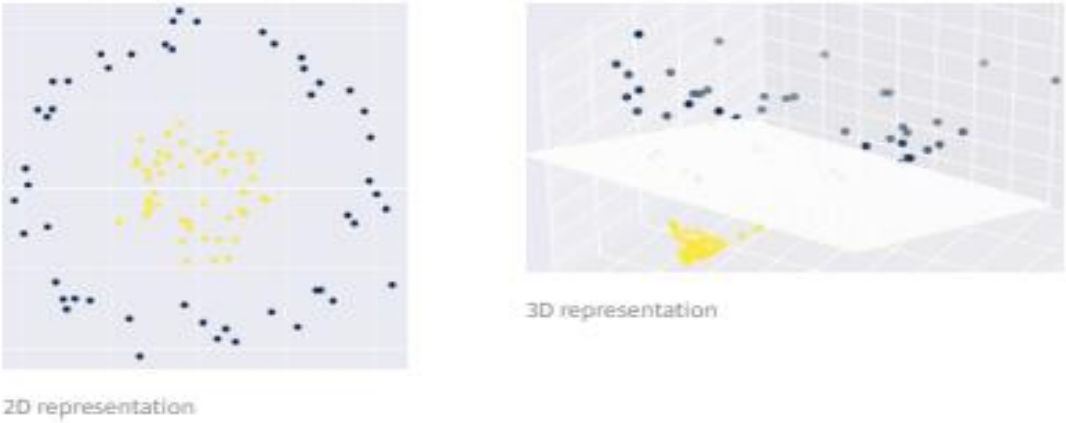
## Role of the Kernel

Let’s assume we have two vectors, x and x*, in a 2D space, and we want to perform a dot product between them to find a linear classifier. But unfortunately, the data is not linearly separable in our current 2D vector space.

To solve this problem, we can map the two vectors to a 3D space.

$$x \rightarrow \phi(x)$$
$$x* \rightarrow \phi(x*)$$

Where ϕ(x) and ϕ(x*) are 3D representations of x and x*.




Now we can perform the dot product between φ(x) and φ(x*) to find our linear classifier in 3D space and then map back to the 2D space.