UNIT - 3

1. Decision Tree Learning

Decision tree learning is a method for approximating discrete-valued target functions, in which the learned function is represented by a decision tree. Learned trees can also be re-represented assets of if-then rules to improve human readability.

A decision tree is a flowchart-like representation of data that graphically resembles a tree that has been drawn upside down. In this analogy, the root of the tree is a decision that has to be made, the tree's branches are actions that can be taken and the tree's leaves are potential decisionoutcomes.

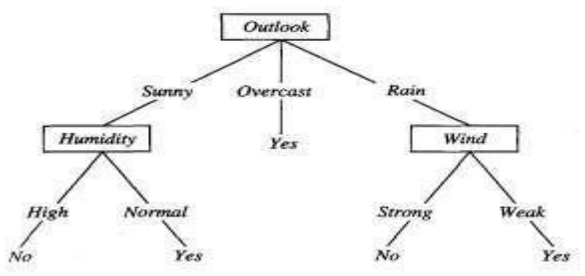
The purpose of a decision tree is to partition a large dataset into subsets that contain instanceswith similar values in order to understand the likely outcomes of specific options.

In machine learning (ML), decision trees are used to predict the class or value of target variablesin supervised learning (SL) regression and classification algorithms. Regression algorithms, alsocalled continuous algorithms, use training data to predict all the future values of a specific data instance within a given period of time. In contrast, classification algorithms use training data to predict the value of a single data instance at a specific moment in time. The sample diagram for the decision tree is shown below:

Fig.

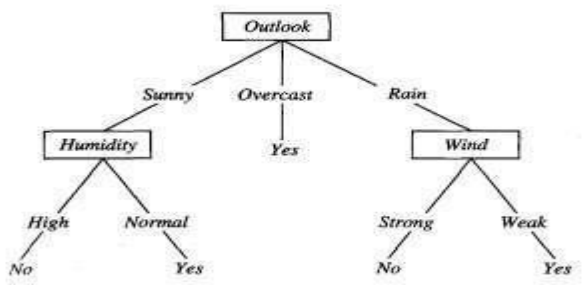
Terminologies of Decision tree

- **Root node:** The base of the decision tree.
- **Splitting:** The process of dividing a node into multiple sub-nodes.
- **Decision node:** When a sub-node is further split into additional sub-nodes.
- **Leaf node:** When a sub-node does not further split into additional sub-nodes; represents possible outcomes.
- **Pruning:** The process of removing sub-nodes of a decision tree.
- **Branch:** A subsection of the decision tree consisting of multiple nodes.



Decision Tree Representation

- Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance.
- Each node in the tree specifies a test of some attribute of the instance, and each branch descending from that node corresponds to one of the possible values for this attribute.
- An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute in the given example.
- This process is then repeated for the subtree rooted at the new node.



- Figure *illustrates* a typical learned decision tree. This decision tree classifies Saturday mornings according to whether they are suitable for playing tennis.
  For example, the instance
  (Outlook = Sunny, Temperature = Hot, Humidity = High, Wind = Strong)
  would be sorted down the left most branch of this decision tree and would therefore be classified as a negative instance
- In general, decision trees represent a disjunction of conjunctions of constraints on the attribute values of instances.
- Each path from the tree root to a leaf corresponds to a conjunction of attribute tests, and the tree itself to a disjunction of these conjunctions.
- For example, the decision tree shown in Figure corresponds to the expression *(Outlook = Sunny* A *Humidity = Normal)* V *(Outlook = Overcast)* v *(Outlook = Rain A Wind = Weak)*

2. Appropriate Problems of Decision Tree

- Instances are represented by attribute-value pairs. Instances are described by a fixed set of attributes (e.g., Temperature) and their values (e.g., Hot). The extensions to the basic algorithm allow handling real-valued attributes as well.

- The target function has discrete output values. It easily extend to learning functions with more than two possible output values.
- Disjunctive descriptions may be required. As noted above, decision trees naturally represent disjunctive expressions.
- The training data may contain errors. Decision tree learning methods are robust to errors, both errors in classifications of the training examples and errors in the attribute values that describe these examples.
- The training data may contain missing attribute values. Decision tree methods can be used even when some training examples have unknown values

3. CART algorithm

This algorithm can be used for both classification & regression. CART algorithm uses Gini Index criterion to split a node to a sub-node. It start with the training set as a root node, after successfully splitting the root node in two, it splits the subsets using the same logic & again split the sub-subsets, recursively until it finds further splitting will not give any pure sub-nodes or maximum number of leaves in a growing tree or termed it as a Tree pruning.

How to calculate Gini Index?

$$GI = \sum_{i=0}^{c} P_i(1 - P_i)$$

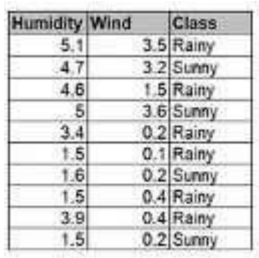
Which can be written as:

$$GI = 1 - \sum_{i=0}^{c} P_i^2$$

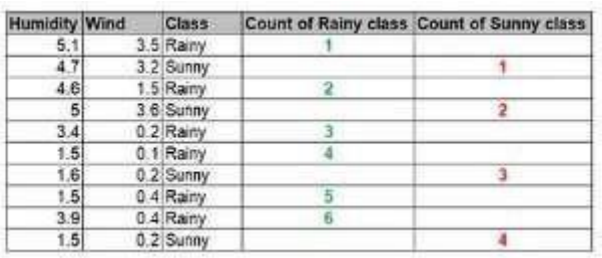
In Gini Index, P is the probability of class **i** & there is total **c** classes.

Considering you have only two predictor/attributes: Humidity & Wind

Class: Rainy & Sunny



Humidity	Wind	Class
5.1	3.5	Rainy
4.7	3.2	Sunny
4.6	1.5	Rainy
5	3.6	Sunny
3.4	0.2	Rainy
1.5	0.1	Rainy
1.6	0.2	Sunny
1.5	0.4	Rainy
3.9	0.4	Rainy
1.5	0.2	Sunny



Humidity	Wind	Class	Count of Rainy class	Count of Sunny class
5.1	3.5	Rainy	1	
4.7	3.2	Sunny		1
4.6	1.5	Rainy	2	
5	3.6	Sunny		2
3.4	0.2	Rainy	3	
1.5	0.1	Rainy	4	
1.6	0.2	Sunny		3
1.5	0.4	Rainy	5	
3.9	0.4	Rainy	6	
1.5	0.2	Sunny		4

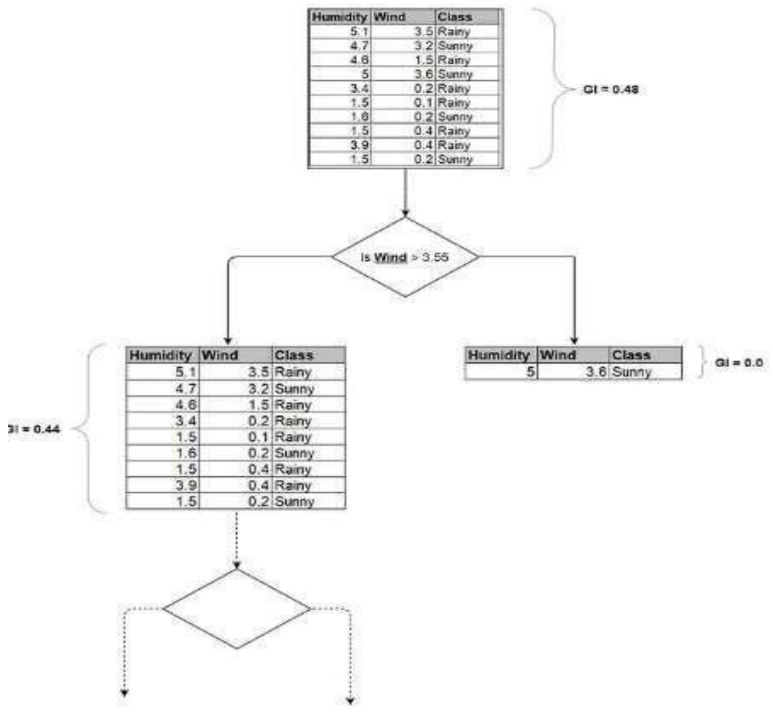
Image: Data & it's distribution by class

GI = 1 — ((num of observations from Feature_1/total observation)^2 + (num of observations from Feature_2/total observation)^2)

GI = 1-((6/10)^2 + (4/10)^2) => 1-(0.36+0.16) => 1–0.52 => 0.48

So, the Gini index for the first/initial set is 0.48

Basic idea on how the Node split happens:

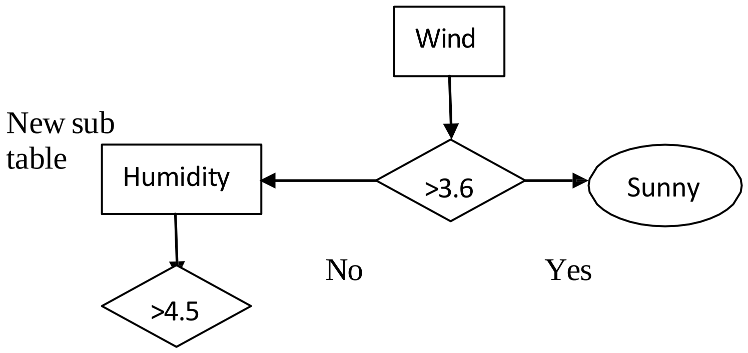


Humidity	Wind	Class
5.1	3.5	Rainy
4.7	3.2	Sunny
4.6	1.5	Rainy
5	3.6	Sunny
3.4	0.2	Rainy
1.5	0.1	Rainy
1.6	0.2	Sunny
1.5	0.4	Rainy
3.9	0.4	Rainy
1.5	0.2	Sunny

Humidity	Wind	Class
5.1	3.5	Rainy
4.7	3.2	Sunny
4.6	1.5	Rainy
3.4	0.2	Rainy
1.5	0.1	Rainy
1.6	0.2	Sunny
1.5	0.4	Rainy
3.9	0.4	Rainy
1.5	0.2	Sunny

Humidity	Wind	Class
5	3.6	Sunny

Also written as,



Based on attribute ―wind‖ (f) & threshold value ―3.55‖ (t) the CART algorithm created nodes/subsets which would give a pure subsets to right side of the above flow

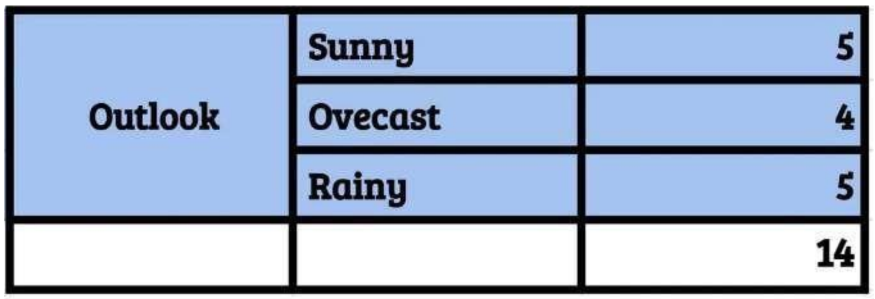
4. **Constructing ID3**

- 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.

5. **<u>C4.5 algorithm</u>**

C4.5 uses the Gain Ratio as the *goodness function* to split the dataset, unlike ID3 which used the Information Gain. The Information Gain function tends to prefer the features with more categories as they tend to have lower entropy. This results in overfitting of the training data. Gain Ratio mitigates this issue by penalising features for having a more categories using a formula called Split Information or Intrinsic Information.

Consider the calculation of Split Information for the Outlook feature.



Outlook	Sunny	5
	Ovecast	4
	Rainy	5
		14

$$\text{Split Information} = \sum -\text{ratio} \times \log_2 \text{ratio} \; \forall \; \text{category} \in \text{feature}$$

$$\text{Split Information(Outlook)} = -\frac{5}{14} \times \log_2 \frac{5}{14} - \frac{4}{14} \times \log_2 \frac{4}{14} - \frac{5}{14} \times \log_2 \frac{5}{14}$$
$$= 1.577$$

Gain Ratio

The notion of Gain introduced earlier tends to favor attributes that have a large number of values. For example, if we have an attribute D that has a distinct value for each record, then Info(D,T) is 0, thus Gain(D,T) is maximal. To compensate for this Quinlan suggests using the following ratio instead of Gain:

$$GainRatio(D,T) = \frac{Gain(D,T)}{SplitInfo(D,T)}$$

where SplitInfo(D,T) is the information due to the split of T on the basis

of the value of the categorical attribute D.

$$SplitInfo(A) = -\sum |Dj|/|D| \text{ x } \log|Dj|/|D|$$

Pseudocode

1. Check **for** the above **base** cases.
2. For each attribute a, find the normalised information gain ratio **from** splitting on a.
3. Let a_best be the attribute **with** the highest normalized information gain.
4. Create a decision node that splits on a_best.
5. Recur on the sublists obtained **by** splitting on a_best, **and** add those nodes **as** children of node.

Advantages of C4.5 over other Decision Tree systems:

1. The algorithm inherently employs Single Pass Pruning Process to Mitigate overfitting.
2. It can work with both **Discrete** and **Continuous** Data
3. C4.5 can handle the issue of incomplete data very well

Improvements in C4.5 over ID3:

· Handling both continuous and discrete

· Handling training data with missing attribute values

· Handling attributes with differing costs.

· Pruning trees after creation

Limitations:

The limitations of C4. 5 is **its information entropy**, it gives poor results for larger distinct attributes

6. Advantages

- Works for numerical or categorical data and variables.
- Models problems with multiple outputs.
- Tests the reliability of the tree.
- Requires less data cleaning than other data modeling techniques.
- Easy to explain to those without an analytical background.

7. Disadvantages:

- Affected by noise in the data.
- Not ideal for large datasets.
- Can disproportionately value, or weigh, attributes.
- The decisions at nodes are limited to binary outcomes, reducing the complexity that the treecan handle.
- Trees can become very complex when dealing with uncertainty and numerous linkedoutcomes.

8. Issues in Decision tree algorithm

- Avoiding overfitting the data
  - Determining how deeply to grow a decision tree.
- Reduced error pruning.
- Rule post-pruning.
- Handling continuous attributes.
  - e.g., temperature
- Choosing an appropriate attribute selection measure.
- Handling training data with missing attribute values.
- Handling attributes with differing costs.
- Improving computational efficiency.

Truncation

It stops the tree while it is still growing so that it may not end up with leaves containing very few data points. Truncation is also known as pre-pruning.

There are various methods that can be used to control the size of the tree or Truncation. Some of the methods are:

- Limit the minimum size of the partition after the split: We can limit the number of data that a leaf should have to split.
- Minimize change in the measure of homogeneity: If homogeneity measure doesn't change a lot we don't split further.
- Limit the depth of the tree
- Set a minimum threshold on the number of samples that appears in the leaf
- Set the limit on the maximum number of leaves present in the tree.

Pruning

Let the tree grow to any complexity. Then, cut the branches of the tree in a bottom-up fashion, starting from the leaves. It is more common to use pruning strategies to avoid overfitting in practical implementations.

Chopping off the tree branches results in a decrease in tree complexity. And definitely helps to overcome overfitting. But how to decide if a branch should be pruned or not?

We divide the dataset into three parts, Training, Validation and Test dataset. We prune the model and calculate the accuracy before pruning and after pruning. If the accuracy of the pruned tree is higher than the accuracy of the original tree on the validation set, then we keep that branch chopped.

Support Vector Machine: Linear SVM Classification, Nonlinear SVM Classification,SVM Regression

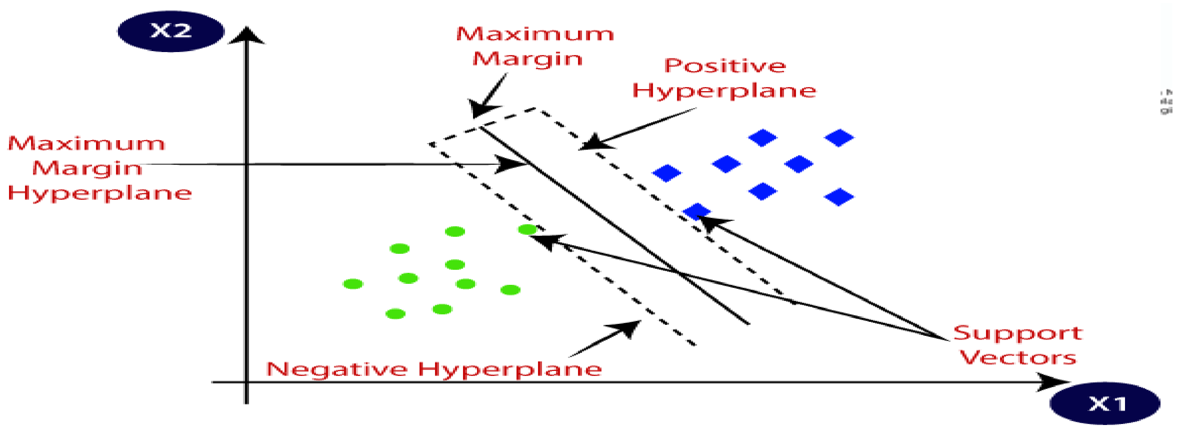
Support Vector Machine Algorithm:

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.

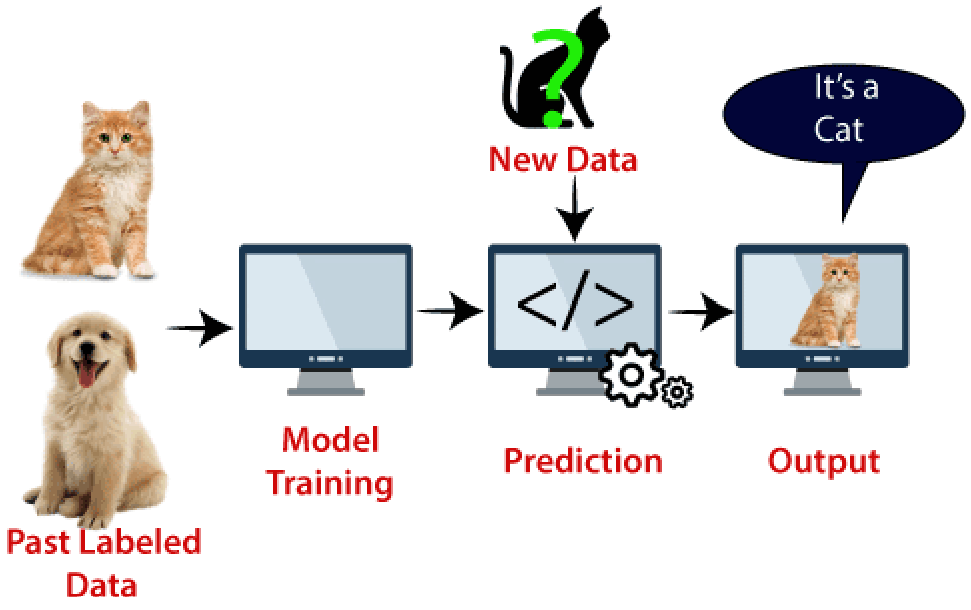
The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.

Consider the below diagram in which there are two different categories that are classified using a decision boundary or 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:Fullscreen



SVM algorithm can be used for **Face detection, image classification, text categorization,** etc.

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 and Support Vectors in the SVM algorithm:

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.

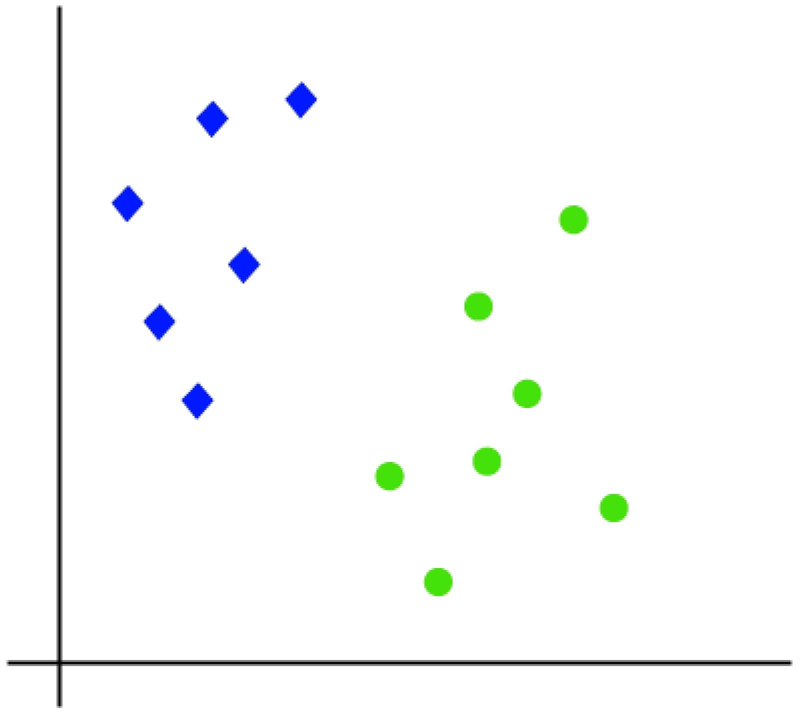
The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.

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.

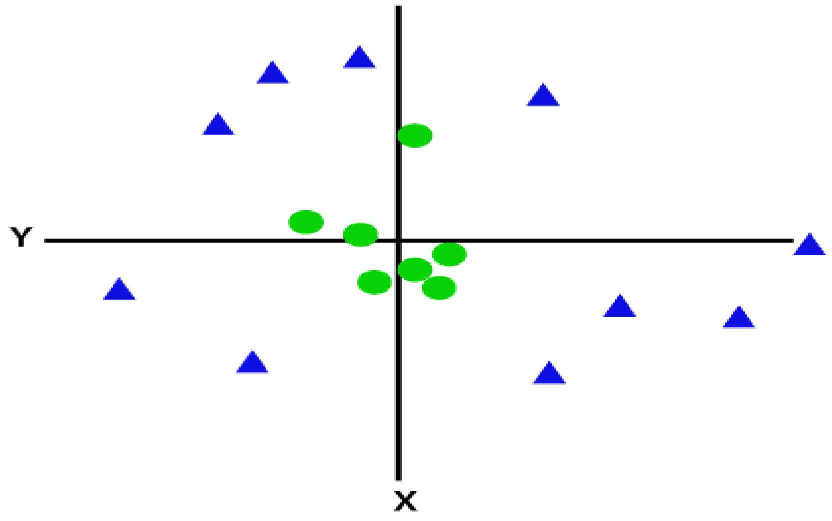
Linear SVM:

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:



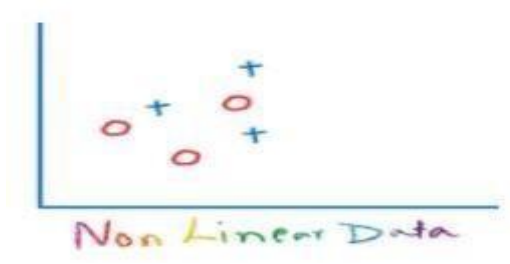
Non-Linear SVM:

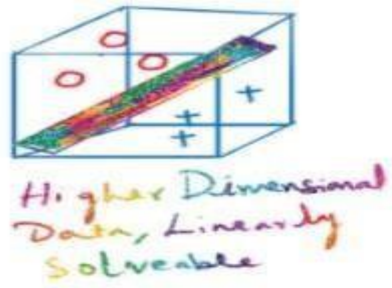
If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:



Support Vector Regression

It is one of the classic examples of supervised Machine learning technique. We could say it’s one of the more powerful models which can be used in classification problems or assigning classes when the data is not linearly separable. I would give a classic kitchen example; I am sure most of us love chips? Of course, I do. I wanted to make homemade chips. I bought potatoes from the vegetable market and hit my kitchen. All I did was follow a YouTube video. I started slicing the potatoes in the hot oil, the result was a disaster, and I ended up getting chips which were dark brown/black.
What was wrong here? I had purchased potatoes, followed the procedure shown in the video and also maintained the right temperature of the oil, did I miss something? When I did my research by asking experts, I found that I did miss the trick, I had chosen potatoes with higher starch content instead of lower ones. All the potatoes looked the same for me, this is where the years and years of training comes into the picture, the experts had well-trained eyes on how to choose potatoes with lower starch. It has some specific features such as these potatoes would look fresh and will have some additional skin which could be peeled from our fingernails, and they look muddy.
With the chips example, I was only trying to tell you about the nonlinear dataset. While many classifiers exist that can classify linearly separable data like logistic regression or linear regression, SVMs can handle highly non-linear data using an amazing technique called kernel trick. It implicitly maps the input vectors to higher dimensional (adds more dimensions) feature spaces by the transformation which rearranges the dataset in such a way that it is linearly solvable.

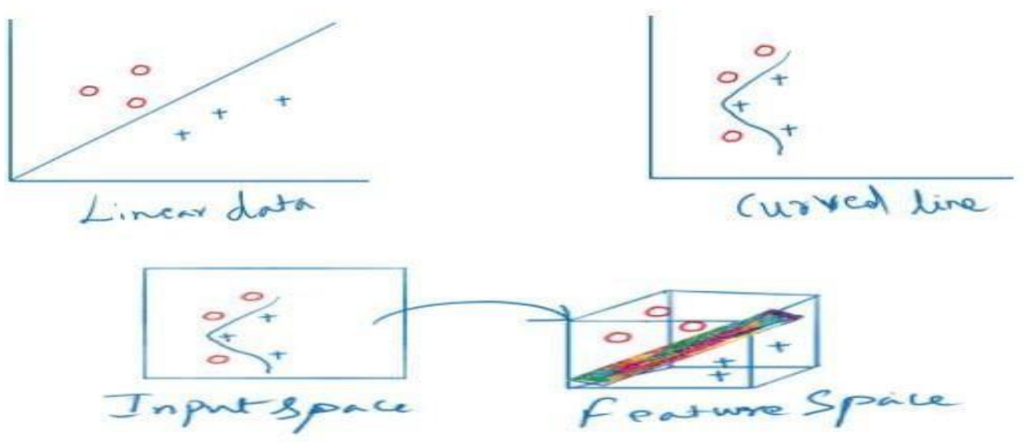




In short, a kernel is a function which places a low dimensional plane to a higher dimensional space where it can be segmented using a plane. In other words, it transforms linearly inseparabledata to separable data by adding more dimensions to it.

There are three kernels which SVM uses the most

- **Linear kernel:** Dot product between two given observations
- **Polynomial kernel:** This allows curved lines in the input space
- **Radial Basis Function (RBF):** It creates complex regions within the feature space



In general, regression problems involve the task of deriving a mapping function which wouldapproximate from input variables to a continuous output variable.

Support Vector Regression uses the same principle of Support Vector Machines. In other words, the approach of using SVMs to solve regression problems is called Support Vector Regression or