ML Unit 4

1. What is decision tree? State the advantages and limitations.

A decision tree is a flowchart-like tree structure where each internal node denotes the feature, branches denote the rules and the leaf nodes denote the result of the algorithm. It is a versatile supervised machine-learning algorithm, which is used for both classification and regression problems. It is one of the very powerful algorithms. And it is also used in Random Forest to train on different subsets of training data, which makes random forest one of the most powerful algorithms in machine learning.

Decision Tree Terminologies

Some of the common Terminologies used in Decision Trees are as follows:

- **Root Node:** It is the topmost node in the tree, which represents the complete dataset. It is the starting point of the decision-making process.
- Decision/Internal Node: A node that symbolizes a choice regarding an input feature. Branching off of internal nodes connects them to leaf nodes or other internal nodes.
- **Leaf/Terminal Node:** A node without any child nodes that indicates a class label or a numerical value.
- **Splitting:** The process of splitting a node into two or more sub-nodes using a split criterion and a selected feature.
- **Branch/Sub-Tree:** A subsection of the decision tree starts at an internal node and ends at the leaf nodes.
- **Parent Node:** The node that divides into one or more child nodes.
- **Child Node:** The nodes that emerge when a parent node is split.
- **Impurity**: A measurement of the target variable's homogeneity in a subset of data. It refers to the degree of randomness or uncertainty in a set of examples. The **Gini index** and **entropy** are two commonly used impurity measurements in decision trees for classifications task
- **Variance**: Variance measures how much the predicted and the target variables vary in different samples of a dataset. It is used for regression problems in decision trees. **Mean squared error, Mean Absolute Error, friedman_mse, or Half Poisson deviance** are used to measure the variance for the regression tasks in the decision tree.

- **Information Gain:** Information gain is a measure of the reduction in impurity achieved by splitting a dataset on a particular feature in a decision tree. The splitting criterion is determined by the feature that offers the greatest information gain, It is used to determine the most informative feature to split on at each node of the tree, with the goal of creating pure subsets
- **Pruning**: The process of removing branches from the tree that do not provide any additional information or lead to overfitting.

Advantages of the Decision Tree:

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

Disadvantages of the Decision Tree:

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

2. What is need of Decision tree.

A decision tree is a type of supervised learning algorithm that is commonly used in machine learning to model and predict outcomes based on input data. It is a tree-like structure where each internal node tests on attribute, each branch corresponds to attribute value and each leaf node represents the final decision or prediction. The decision tree algorithm falls under the category of supervised learning. They can be used to solve both **regression** and **classification problems**.

Why Decision Tree?

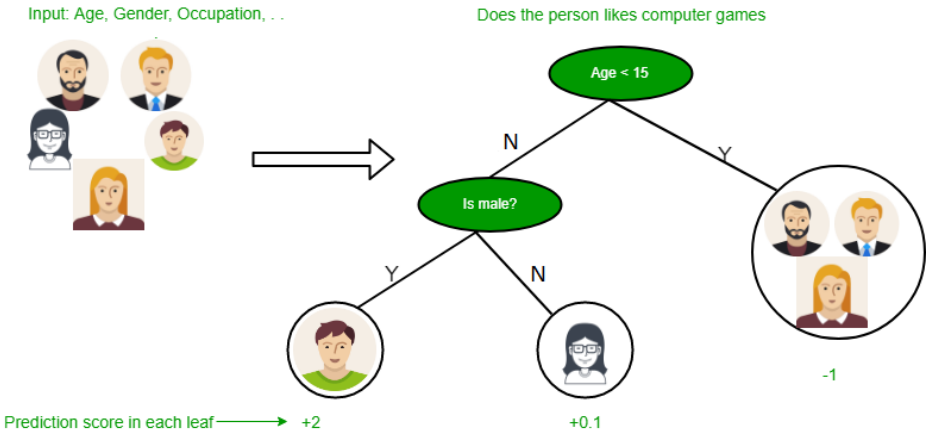
Decision trees are widely used in machine learning for a number of reasons:

- Decision trees are so versatile in simulating intricate decision-making processes, because of their interpretability and versatility.

- Their portrayal of complex choice scenarios that take into account a variety of causes and outcomes is made possible by their hierarchical structure.
- They provide comprehensible insights into the decision logic, decision trees are especially helpful for tasks involving categorisation and regression.
- They are proficient with both numerical and categorical data, and they can easily adapt to a variety of datasets thanks to their autonomous feature selection capability.
- Decision trees also provide simple visualization, which helps to comprehend and elucidate the underlying decision processes in a model.

Decision Tree Approach

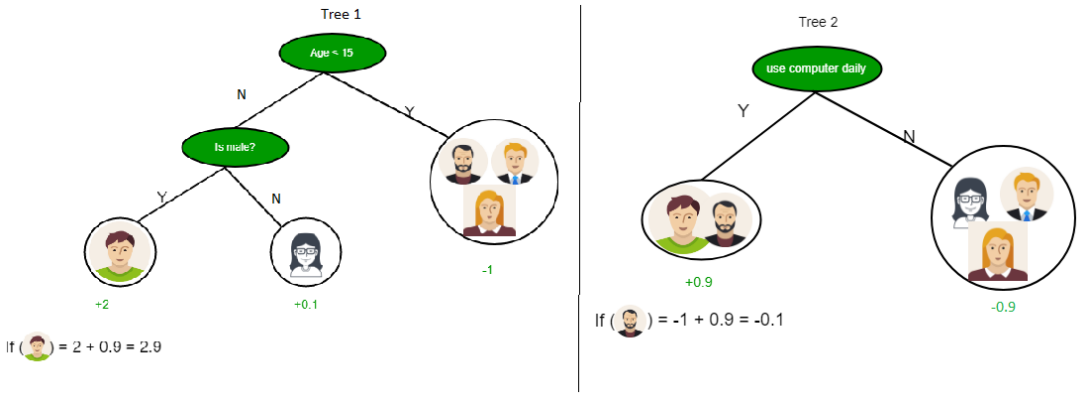
Decision tree uses the tree representation to solve the problem in which each leaf node corresponds to a class label and attributes are represented on the internal node of the tree. We can represent any boolean function on discrete attributes using the decision tree.



Below are some assumptions that we made while using the decision tree:

At the beginning, we consider the whole training set as the root.

- Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
- On the basis of attribute values, records are distributed recursively.
- We use statistical methods for ordering attributes as root or the internal node.

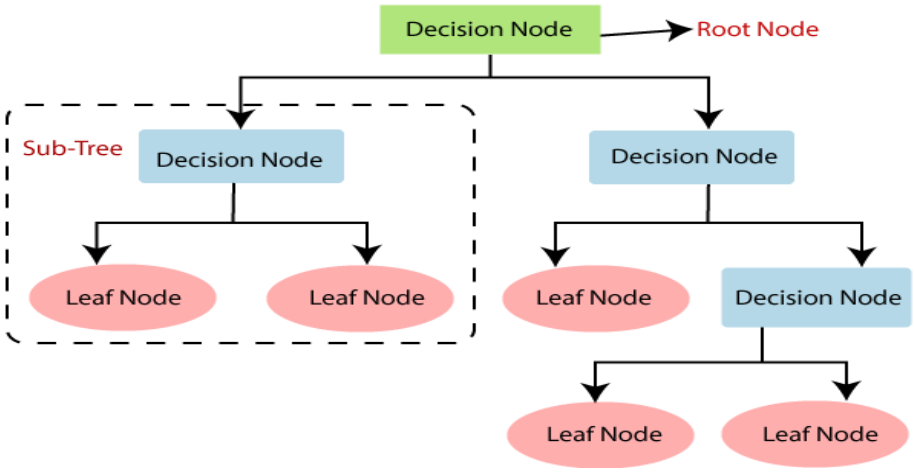


As you can see from the above image the Decision Tree works on the Sum of Product form which is also known as *Disjunctive Normal Form*. In the above image, we are predicting the use of computer in the daily life of people. In the Decision Tree, the major challenge is the identification of the attribute for the root node at each level. This process is known as attribute selection.

3. Explain 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.
- The decisions or the test are performed on the basis of features of the given dataset.
- 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.
- Below diagram explains the general structure of a decision tree:



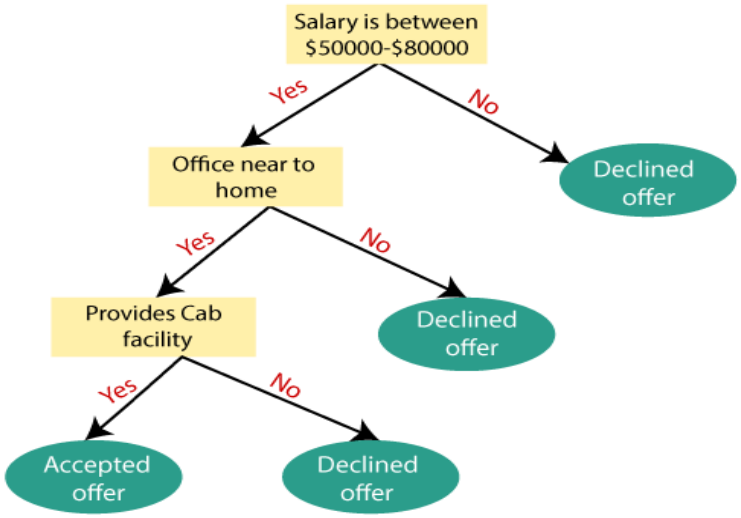
How does the Decision Tree algorithm Work?

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.
- **Step-2:** Find the best attribute in the dataset using **Attribute Selection Measure (ASM).**
- **Step-3:** Divide the S into subsets that contains possible values for the best attributes.
- **Step-4:** Generate the decision tree node, which contains the best attribute.
- **Step-5:** Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Example: Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not. So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM). The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels. The next decision node further gets split into one decision node (Cab facility) and one leaf node. Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer). Consider the below diagram:



4. What is information gain and entropy in decision tree?

In machine learning, specifically in the context of decision trees, information gain and entropy are key concepts used for splitting data to make decisions.

1. Entropy: Entropy measures the impurity or uncertainty in a dataset. In the context of a decision tree, it represents the randomness in the data. Mathematically, for a binary classification problem, entropy is calculated as:

   $Entropy(S) = -p1\log_2(p1) - p2\log_2(p2)$

   Where:

- S is the dataset.
- $p1$ is the probability of one class.
- $p2$ is the probability of the other class.

Entropy is maximum when the dataset contains an equal proportion of both classes (maximum uncertainty), and it decreases as the dataset becomes more homogeneous.

- It is the measure of impurity (or) uncertainty in the data. It lies between 0 to 1 and is calculated using the below formula.
- **For example,** let's see the formula for a use case with 2 classes- Yes and No.
- $Entropy(s) = -P(Yes)log2P(Yes) - P(No)log2P(No)$.
- Here **Yes** and **No** are two classes and s is the total number of samples. P(Yes) is the probability of Yes and P(No) is the probability of No.
- If there are many classes, then the formula is
- $-P1log2P1 - P2log2P2 - ..... - Pnlog2Pn$
- Here n is the number of classes.
- The entropy of each feature after every split is calculated and the splitting is continued. The lesser the entropy the better will be the model because the classes would be split better because of less uncertainty.

2. Information Gain: Information gain measures the effectiveness of a particular attribute in reducing uncertainty (entropy) in the dataset. It helps in deciding the order of attributes in the nodes of a decision tree. Information gain is calculated as the difference between the entropy of the parent node and the weighted sum of the entropies of the child nodes resulting from splitting the data based on a particular attribute.

$Gain(S,A) = Entropy(S) - \sum |Sv||S| \cdot Entropy(Sv) Gain(S,A) = Entropy(S) - \sum |S||Sv| \cdot Entropy(Sv)$

Where:

- SS is the dataset.
- AA is the attribute being considered for splitting.
- $SvSv$ is the subset of SS for which attribute AA has value vv.
- $|S||S|$ is the total number of instances in the dataset.

The attribute with the highest information gain is chosen as the splitting attribute for a node in the decision tree.

In summary, entropy measures the impurity or randomness in the data, and information gain measures the reduction in entropy achieved by splitting the data based on a particular attribute. These concepts are fundamental in the construction of decision trees for classification tasks in machine learning.

- Information gain is simply the measure of change in entropy. The higher the information gain, the lower is the entropy. Thus, for a model to be good, it should have high Information gain. A decision tree algorithm in general tries to maximize the value of information gain, and an attribute/feature having the highest information gain is split first.
- *Information Gain=EntropyBeforeSplit−EntropyAfterSplit*
-
- Where "before" is the dataset before the split, N is the number of subsets that got generated after we split the node, and (i, after) is the subset 'i' after the split.

5. What are algorithms are used in decision tree.

The different decision tree algorithms are listed below:

ID3(Iterative Dichotomiser 3)

C4.5

CART(Classification and Regression Trees)

CHAID (Chi-Square Automatic Interaction Detection)

MARS(Multivariate Adaptive Regression Splines)

ID3 (Iterative Dichotomiser 3)

An approach for decision trees called ID3 (Iterative Dichotomiser 3) is employed in classification applications. It is one of the first and most used decision tree algorithms, created by Ross Quinlan in 1986. The ID3 algorithm builds a decision tree from a given dataset using a greedy, top-down methodology.

It works by greedily choosing the feature that maximizes the information gain at each node. ID3 calculates entropy and information gain for each feature and selects the feature with the highest information gain for splitting.

ID3 uses entropy to measure the uncertainty or disorder in a dataset. Entropy, denoted by H(D) for dataset D, is calculated using the formula:

H(D) = \Sigma^n _{i=1}\;p_{i}\; log_{2}(p_{i})

Information gain quantifies the reduction in entropy achieved by splitting the data based on a particular feature. Features with higher information gain are preferred for splitting. Information gain is calculated as follows:

Information\; Gain = H(D) - \Sigma^V_{v=1} \frac{|D_{v}|}{|D|}H (D_{v})
Every decision tree node's dataset is recursively divided using the ID3 algorithm according to the chosen attribute. This method keeps going until either there are no more attributes to divide on, or all the examples in a node belong to the same class.

The decision tree may be trimmed after it is constructed in order to enhance generalization and lessen overfitting. In order to do this, nodes that do not considerably improve the correctness of the tree must be removed.

A couple of the ID3 algorithm's drawbacks are that it tends to overfit the training set and cannot directly handle continuous attributes. Owing to these drawbacks, other decision tree algorithms that address some of these problems have been developed, including C4.5 and CART.

Entropy, information gain, and recursive partitioning are three key principles in the ID3 algorithm, which is a fundamental technique for creating decision trees. Mastering these ideas is crucial to learning about decision tree algorithms in machine learning.

C4.5

As an enhancement to the ID3 algorithm, Ross Quinlan created the decision tree algorithm C4.5. In machine learning and data mining applications, it is a well-liked approach for

creating decision trees. Certain drawbacks of the ID3 algorithm are addressed in C4.5, including its incapacity to deal with continuous characteristics and propensity to overfit the training set.

A modification of information gain known as the gain ratio is used to address the bias towards qualities with many values. It is computed by dividing the information gain by the intrinsic information, which is a measurement of the quantity of data required to characterize an attribute's values.

Gain Ratio = \frac{Split\; gain}{Gain\;\;information}

Where Split Information represents the entropy of the feature itself. The feature with the highest gain ratio is chosen for splitting.

When dealing with continuous attributes, C4.5 sorts the attribute's values first, and then chooses the midpoint between each pair of adjacent values as a potential split point. Next, it determines which split point has the largest value by calculating the information gain or gain ratio for each.

By turning every path from the root to a leaf into a rule, C4.5 can also produce rules from the decision tree. Predictions based on fresh data can be generated using the rules.

C4.5 is an effective technique for creating decision trees that can produce rules from the tree and handle both discrete and continuous attributes. The model's accuracy is increased and overfitting is prevented by its utilization of gain ratio and decreased error pruning. Nevertheless, it might still be susceptible to noisy data and might not function effectively on datasets with a lot of features.

CART (Classification and Regression Trees)

CART is a decision tree algorithm that can be used for both classification and regression tasks. It works by finding splits that minimize the Gini impurity, a measure of impurity in

the data. CART uses Gini impurity for classification. When selecting a feature to split, it calculates the Gini impurity for each possible split and chooses the one with the lowest impurity.

The likelihood of incorrectly classifying an element selected at random and labeled in accordance with the distribution of labels in the set is measured by the Gini impurity.

Gini Impurity (for Classification) :CART uses Gini impurity as the criterion to measure the impurity or purity of a dataset. Gini impurity, denoted by Gini(D) for dataset D, is calculated using the formula:

Gini(D) = 1 - \Sigma^n _{i=1}\; p^2_{i}

CART can be used to create regression trees for continuous target variables in addition to classification. When deciding which subsets to split, the algorithm in this instance minimizes the variance of the target variable inside each subset.

- **Mean Squared Error (for Regression)**: For regression tasks, CART uses mean squared error (MSE) to evaluate splits. MSE measures the average squared difference between the predicted and actual values. The split with the lowest MSE is chosen.

Where represents the target values, is the mean of the target values in dataset D, and |D| is the number of data points in D.

Recursively dividing the dataset according to the characteristic that minimizes the Gini impurity or maximizes the information gain at each stage is done by CART using a greedy strategy. It looks at all potential split points for each attribute and selects the one that produces the lowest Gini impurity for the subsets that are generated.

To lessen overfitting, CART employs a method known as cost-complexity pruning once the decision tree is constructed. This entails determining the tree that minimizes the total cost, which is the sum of the impurity and the complexity, by adding a complexity parameter to the impurity measure.

Every internal node in a binary tree created by CART has exactly two child nodes. This facilitates the splitting procedure and facilitates the interpretation of the resultant trees.

CHAID (Chi-Square Automatic Interaction Detection)

CHAID is a decision tree algorithm that uses **chi-square tests** to determine the best splits for categorical variables. It works by recursively splitting the data into smaller and smaller subsets until each subset contains only data points of the same class or within a certain range of values. The algorithm selects the feature to split on at each node based on the **chi-squared test of independence**, which is a statistical test that measures the relationship between two variables. In CHAID, the algorithm selects the feature that has the highest chi-squared statistic, which means that it has the strongest relationship with the target variable. It is particularly useful for analyzing large datasets with many categorical variables.

To perform the Chi-Square test, one must compute the Chi-Square statistic, which may be found using the following formula:

$$X^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

Where,

Oi represents the observed frequency and Ei represents the expected frequency in each category. The observed distribution is compared to the expected distribution using the Chi-Square statistic to see if there is a significant difference.

CHAID can be used for both classification and regression tasks. In classification tasks, the algorithm predicts the class label of a new data point by traversing the decision tree from the root node to a leaf node. The class label of the leaf node is then assigned to the new data point.

In regression tasks, CHAID predicts the value of the target variable for a new data point by averaging the values of the target variable at the leaf node where the new data point falls.

MARS (Multivariate Adaptive Regression Splines)

MARS is an extension of CART that uses **splines to model non-linear relationships** between variables. MARS is a regression algorithm that uses a technique called forward stepwise selection to construct a piecewise linear model. A piecewise linear model is a model where the output variable is a linear function of the input variables, but the slope of the linear function can change at different points in the input space. The sites

where piecewise linear functions (basis functions) connect are known as knots.Based on the distribution of the data and the requirement to capture non-linearities, MARS automatically chooses and positions knots

Basis Functions: Basis functions, or piecewise linear functions, are used by MARS to represent the relationship between predictors and the response variable. Simple linear functions that are defined across a particular range of a predictor variable make up each basis function.

In MARS, a basis function is described as:

$$h(x) = \begin{cases} x - t \ \ if \ x > t \\ t - x \ \ if x \leq t \end{cases}$$

Where , x is a predictor variable and t is the knot function.

Knot Function: The sites where piecewise linear functions (basis functions) connect are known as knots. Based on the distribution of the data and the requirement to capture non-linearities, MARS automatically chooses and positions knots

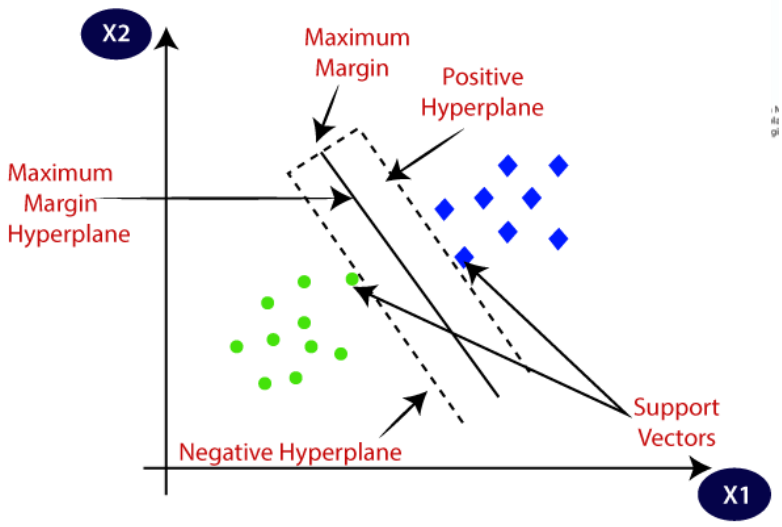
MARS starts by constructing a model with a single piece. The algorithm then uses forward stepwise selection to add new pieces to the model. At each step, the algorithm adds the piece that reduces the residual sum of squares the most. The algorithm continues adding pieces until the model reaches a specified level of complexity. MARS can be used to model complex relationships between variables. It is particularly useful for modeling complex relationships in data.

6. **What is SVM? Explain in Detail.**

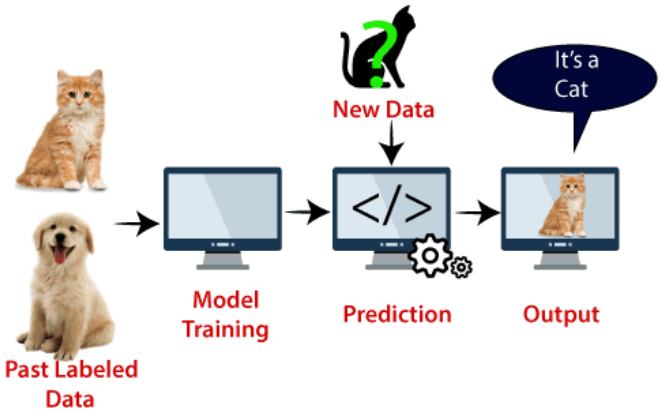
Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

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:

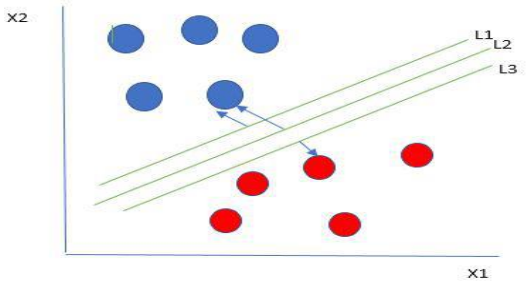


SVM algorithm can be used for **Face detection, image classification, text categorization,** etc.

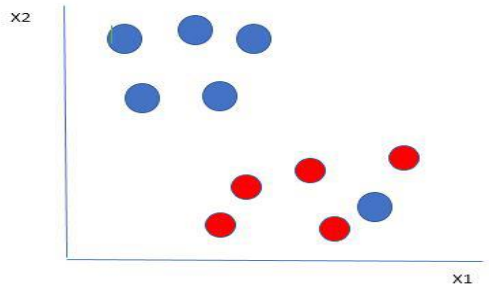
Types of SVM

SVM can be of two types:

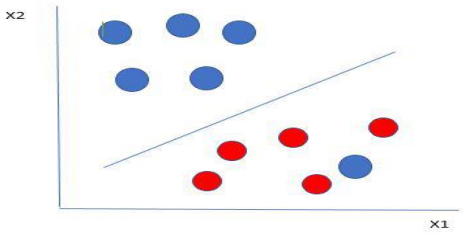
- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
- **Non-linear SVM:** Non-Linear 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.
- **How does the SVM work?**
- One reasonable choice as the best hyperplane is the one that represents the largest separation or margin between the two classes.



- So, we choose the hyperplane whose distance from it to the nearest data point on each side is maximized. If such a hyperplane exists it is known as the **maximum-margin hyperplane/hard margin**. So, from the above figure, we choose L2. Let's consider a scenario like shown below

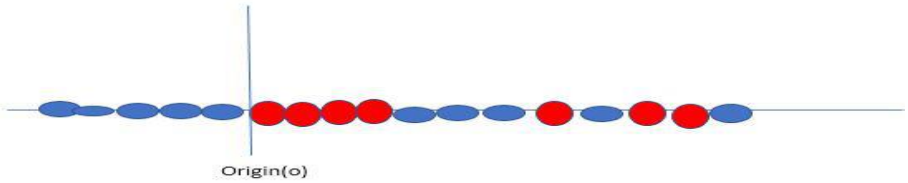


- Here we have one blue ball in the boundary of the red ball. So how does SVM classify the data? It's simple! The blue ball in the boundary of red ones is an outlier of blue balls. The SVM algorithm has the characteristics to ignore the outlier and finds the best hyperplane that maximizes the margin. SVM is robust to outliers.



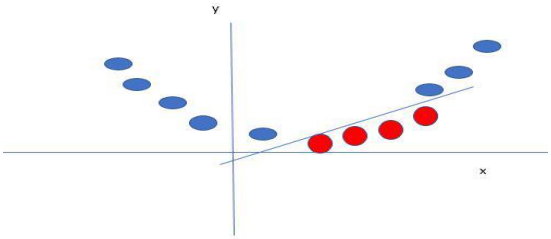
So, in this type of data point what SVM does is, finds the maximum margin as done with previous data sets along with that it adds a penalty each time a point crosses the margin. So, the margins in these types of cases are called soft margins. When there is a soft margin to the data set, the SVM tries to minimize (1/margin+∧(∑penalty)). Hinge loss is a commonly used penalty. If no violations no hinge loss. If violations hinge loss proportional to the distance of violation.

Till now, we were talking about linearly separable data(the group of blue balls and red balls are separable by a straight line/linear line). What to do if data are not linearly separable?



Say, our data is shown in the figure above. SVM solves this by creating a new variable using a kernel. We call a point xi on the line, and we create a new variable Yi as a function of distance from origin o.so if we plot this, we get something like as shown below

In this case, the new variable y is created as a function of distance from the origin. A non-linear function that creates a new variable is referred to as a kernel.



Mapping 1D data to 2D to become able to separate the two classes

7. **Explain Hyperplane and Support Vector in SVM algorithm.**

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.

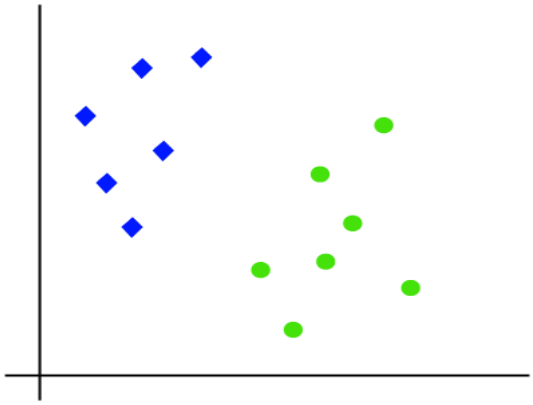
Support Vectors:

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.

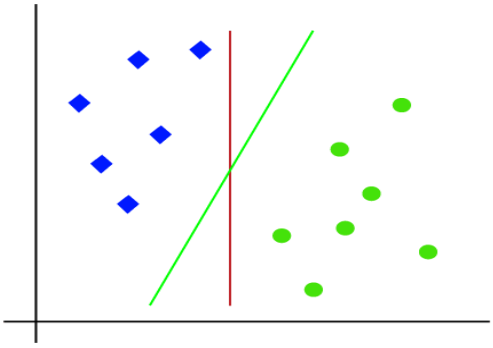
How does SVM works?

Linear SVM:

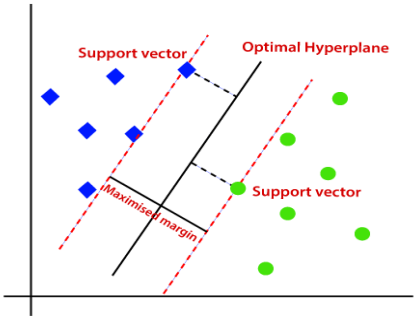
The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image:



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:

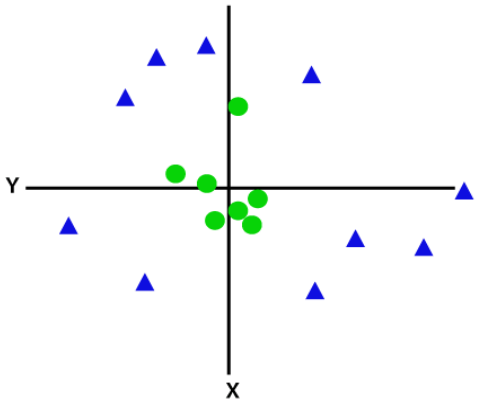


Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane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

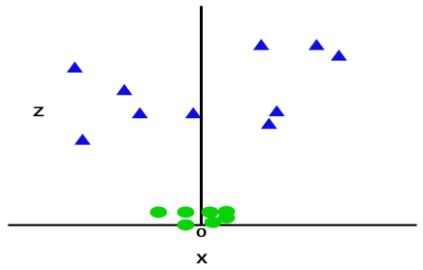
If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:



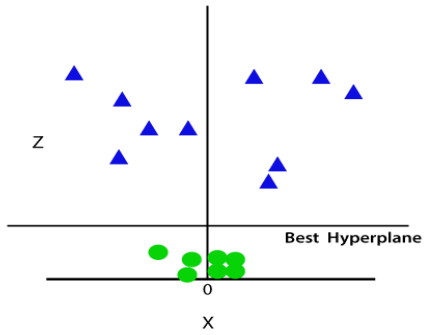
So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

$z=x^2 +y^2$

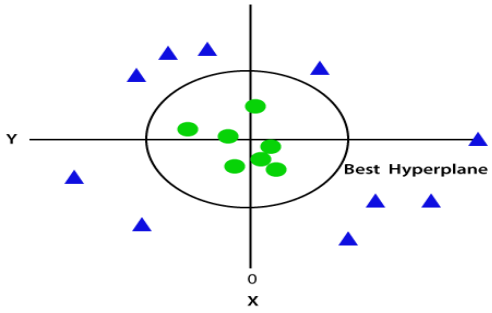
By adding the third dimension, the sample space will become as below image:



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:



Hence we get a circumference of radius 1 in case of non-linear data.

8. **Which are the Pros and cons of SVM Classifier.**

Advantages of support vector machine:

- Support vector machine works comparably well when there is an understandable margin of dissociation between classes.
- It is more productive in high-dimensional spaces.
- It is effective in instances where the number of dimensions is larger than the number of specimens.
- Support vector machine is comparably memory systematic. Support Vector Machine (SVM) is a powerful supervised machine learning algorithm with several advantages. Some of the main advantages of SVM include:
- Handling high-dimensional data: SVMs are effective in handling high-dimensional data, which is common in many applications such as image and text classification.
- Handling small datasets: SVMs can perform well with small datasets, as they only require a small number of support vectors to define the boundary.
- Modeling non-linear decision boundaries: SVMs can model non-linear decision boundaries by using the kernel trick, which maps the data into a higher-dimensional space where the data becomes linearly separable.
- Robustness to noise: SVMs are robust to noise in the data, as the decision boundary is determined by the support vectors, which are the closest data points to the boundary.
- Generalization: SVMs have good generalization performance, which means that they are able to classify new, unseen data well.
- Versatility: SVMs can be used for both classification and regression tasks, and it can be applied to a wide range of applications such as natural language processing, computer vision, and bioinformatics.
- Sparse solution: SVMs have sparse solutions, which means that they only use a subset of the training data to make predictions. This makes the algorithm more efficient and less prone to overfitting.
- Regularization: SVMs can be regularized, which means that the algorithm can be modified to avoid overfitting.

Disadvantages of support vector machine:

- Support vector machine algorithm is not acceptable for large data sets.
- It does not execute very well when the data set has more sound i.e. target classes are overlapping.
- In cases where the number of properties for each data point outstrips the number of training data specimens, the support vector machine will underperform.
- As the support vector classifier works by placing data points, above and below the classifying hyperplane there is no probabilistic clarification for the classification.Support Vector Machine (SVM) is a powerful supervised machine learning algorithm, but it also has some limitations and disadvantages. Some of the main disadvantages of SVM include:
- Computationally expensive: SVMs can be computationally expensive for large datasets, as the algorithm requires solving a quadratic optimization problem.
- Choice of kernel: The choice of kernel can greatly affect the performance of an SVM, and it can be difficult to determine the best kernel for a given dataset.
- Sensitivity to the choice of parameters: SVMs can be sensitive to the choice of parameters, such as the regularization parameter, and it can be difficult to determine the optimal parameter values for a given dataset.
- Memory-intensive: SVMs can be memory-intensive, as the algorithm requires storing the kernel matrix, which can be large for large datasets.
- Limited to two-class problems: SVMs are primarily used for two-class problems, although multi-class problems can be solved by using one-versus-one or one-versus-all strategies.
- Lack of probabilistic interpretation: SVMs do not provide a probabilistic interpretation of the decision boundary, which can be a disadvantage in some applications.
- Not suitable for large datasets with many features: SVMs can be very slow and can consume a lot of memory when the dataset has many features.
- Not suitable for datasets with missing values: SVMs requires complete datasets, with no missing values, it can not handle missing values**.**

9. What is Kernel trick in SVM?