Classification- Decision trees and Support Vector Machine:

DECISION TREE

❑INTRODUCTION :

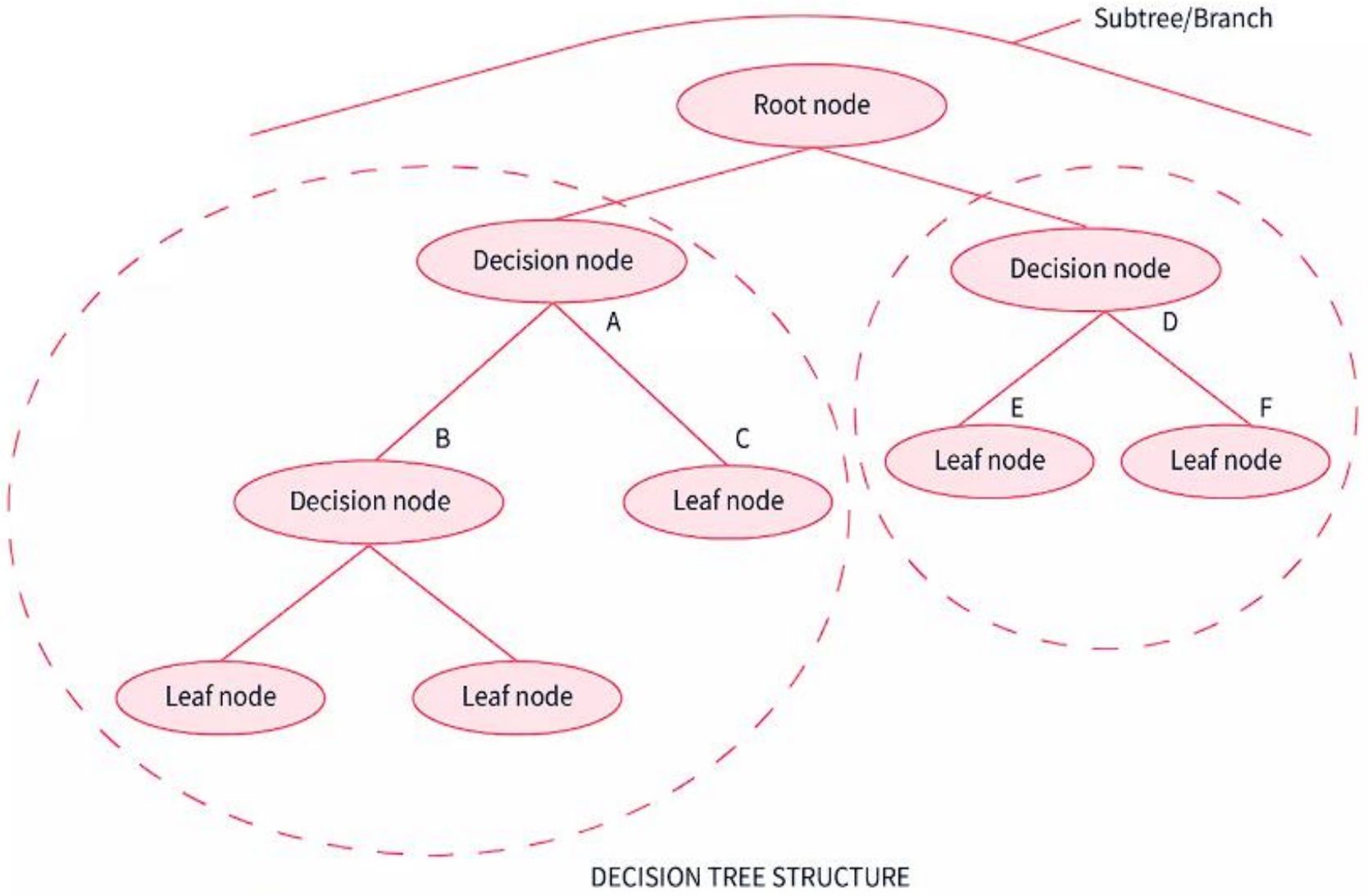
A decision tree is a flowchart-like tree structure where each internal node denotes the feature, branches denote the rules, and the leaf nodes denote the result of the algorithm. It is a versatile supervised machine-learning algorithm, which is used for both classification and regression problems. It 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.

Classification based decision trees.

- Classification is used when the data we have and the data that we are making predictions on are discrete(or)categorical.
- Let's understand the meaning of discrete and continuous data. Ever heard of something like “The system has half defect”? No right!. A system either has a complete defect or no defect at all. There are no halves, quarters, or fractions. This type of data is called Discrete data ex: Yes/no, Good/bad, etc. Only a finite number of values are possible, and the values cannot be subdivided further.
- Continuous data is data that can be measured and broken down into smaller parts and still have meaning. Money, temperature, and time are continuous. **Ex:** If you have many pizzas then counting the no. of pizzas is discrete and measuring each pizza and recording the size (i.e., 25.2 cm ,25.2 cm, 26.1cm,26.1 cm, 29.5 cm,29.5 cm, ? *etc.*) that’s continuous data.

Important Terminologies Related to Decision Trees:

1. **Root Node**: The root node is the very first node in the tree. This node initiates the whole decision-making process by further getting divided into 2 or more sets of nodes and each node consists of a feature in the data set.
2. **Splitting**: A node is broken down into sub-nodes at every level and this process is called splitting.
3. **Decision Node**: When a node can be broken down into 2 or more nodes then it's called a decision node. This breaking down happens according to the number of different decisions that can be made from that node.
4. **Leaf / Terminal Node**: Nodes that cannot be broken down into sub-nodes anymore.
5. **Pruning**: This is the process of removing an unwanted part of a tree. To be precise it can be a node (or) a branch in the tree too.
6. **Branch / Sub-Tree**: When a tree is split into different sub-parts it is known to be a branch in a tree (or) a subtree.
7. **Parent and Child Node**: When a node is divided into sub-nodes the sub-nodes are called the child nodes and the node that got split is called the parent node of all these child nodes.



DECISION TREE STRUCTURE

A is the parent of B, C Child nodes

D is the parent of E, F Child nodes

Why Use Decision Trees?

Here are some of the reasons that state why we should use decision trees while working on machine learning use cases.

1) While using decision trees we get to see the problem more clearly and understand which node is giving rise to what as it forms a tree-like structure. A decision tree almost works like a person doing their day-to-day activities and taking the best decisions related to a certain situation based on various deciding factors.

2) Decision trees help us to see all possible cases that a decision can led to and help us make the decision accordingly.

3) It helps us in mathematically finding out the probabilities of achieving a certain outcome

ADVANTAGES	DISADVANTAGES
1. Decision trees can generate understandable rules.	1. Decision trees are less appropriate for estimation tasks where the goal is to predict the value of a continuous attribute.
2. Decision trees perform classification without requiring much computation.	2. Decision trees are prone to errors in classification problems with many classes and a relatively small number of training examples.
3. Decision trees can handle both continuous and categorical variables.	3. Decision trees can be computationally expensive to train. The process of growing a decision tree is computationally expensive.
4. Decision trees provide a clear indication of which fields are most important for prediction or classification.	4. Decision trees are prone to overfitting the training data, particularly when the tree is very deep or complex.
5. Ease of use: Decision trees are simple to use and don't require a lot of technical expertise, making them accessible to a wide range of users.	5. Small variations in the training data can result in different decision trees being generated, which can be a problem when trying to compare or reproduce results.
6. Scalability: Decision trees can handle large datasets and can be easily parallelized to	6. Many decision tree algorithms do not handle missing data well and require imputation or

Constructing and understanding Decision trees :

❑*How Does Decision Tree Algorithm Work?*

- Let's look at the steps that the decision tree algorithm follows.
- **Step 1:** The first node of the tree is always the root node, and it consists of the whole dataset. Let's refer to the root node as S.
- **Step 2:** Use attribute selection measures to find the best attribute.
- **Step 3:** Now divide the root node (S) into subsets that contain various possible values of the best attribute that you found out in step 2.
- **Step 4:** Now generate a decision tree node that consists of the best attribute.
- **Step 5:** Now make new decision trees using the subsets that were created in step - 3. Do this step recursively till the node cannot be divided anymore i.e. leaf node.

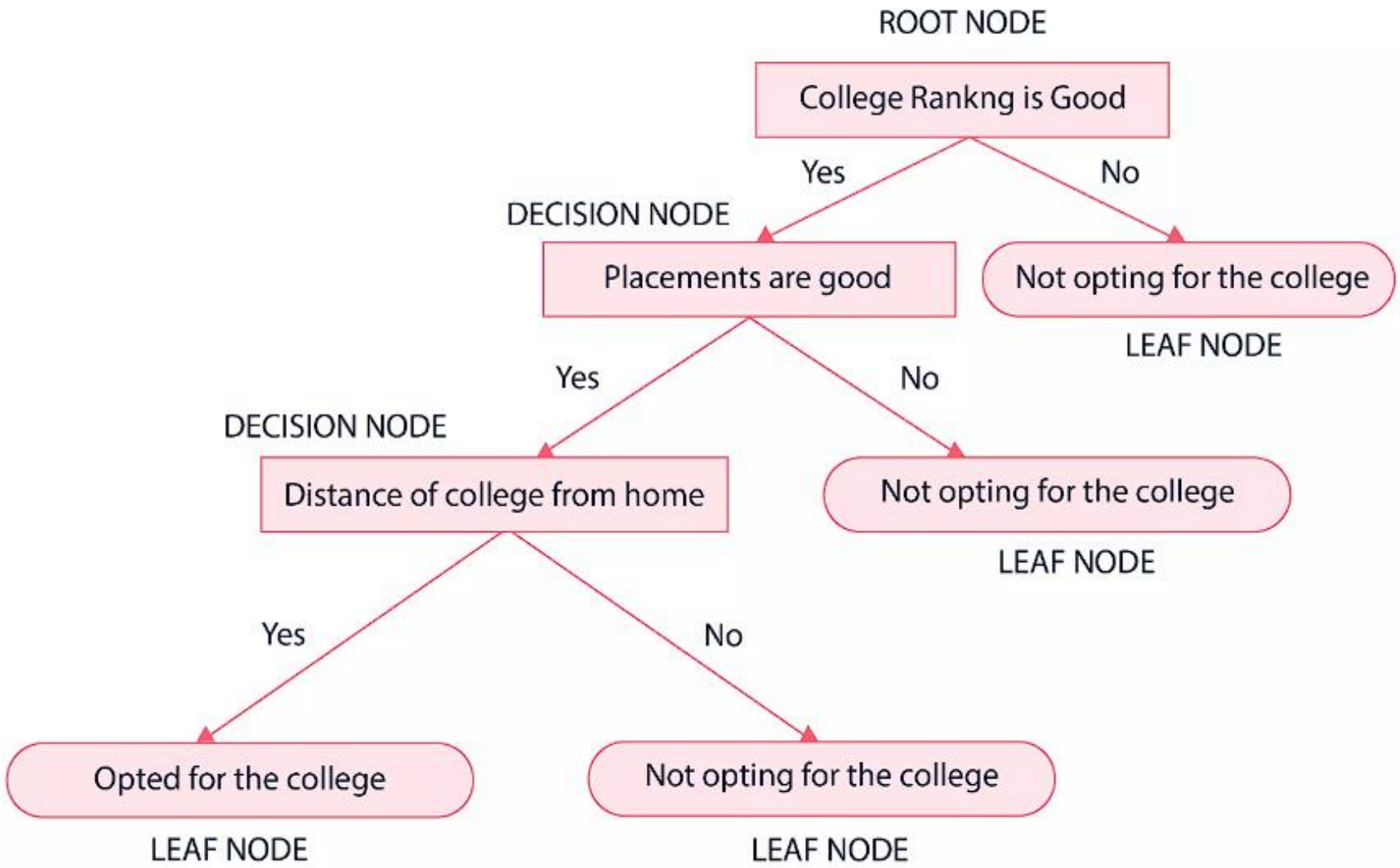
EXAMPLE :

Let's say there is a student who wants to join an engineering college and wants to decide among the various options he/she has.

Let's approach this problem using decision trees. The decision tree starts with the root node (*S*), here let us consider the root node as the college ranking by considering attribute selection measures.

The root node then splits into the next decision node(placements at college) & a leaf node(not opting for the college because the college ranking isn't good).

The next decision node further gets split into one decision node (distance of college from home) and one leaf node (not opting for the college because the placements aren't good). Lastly, the decision node splits into two leaf nodes (Opting for the college and Not opting for the college.). Consider the below diagram.

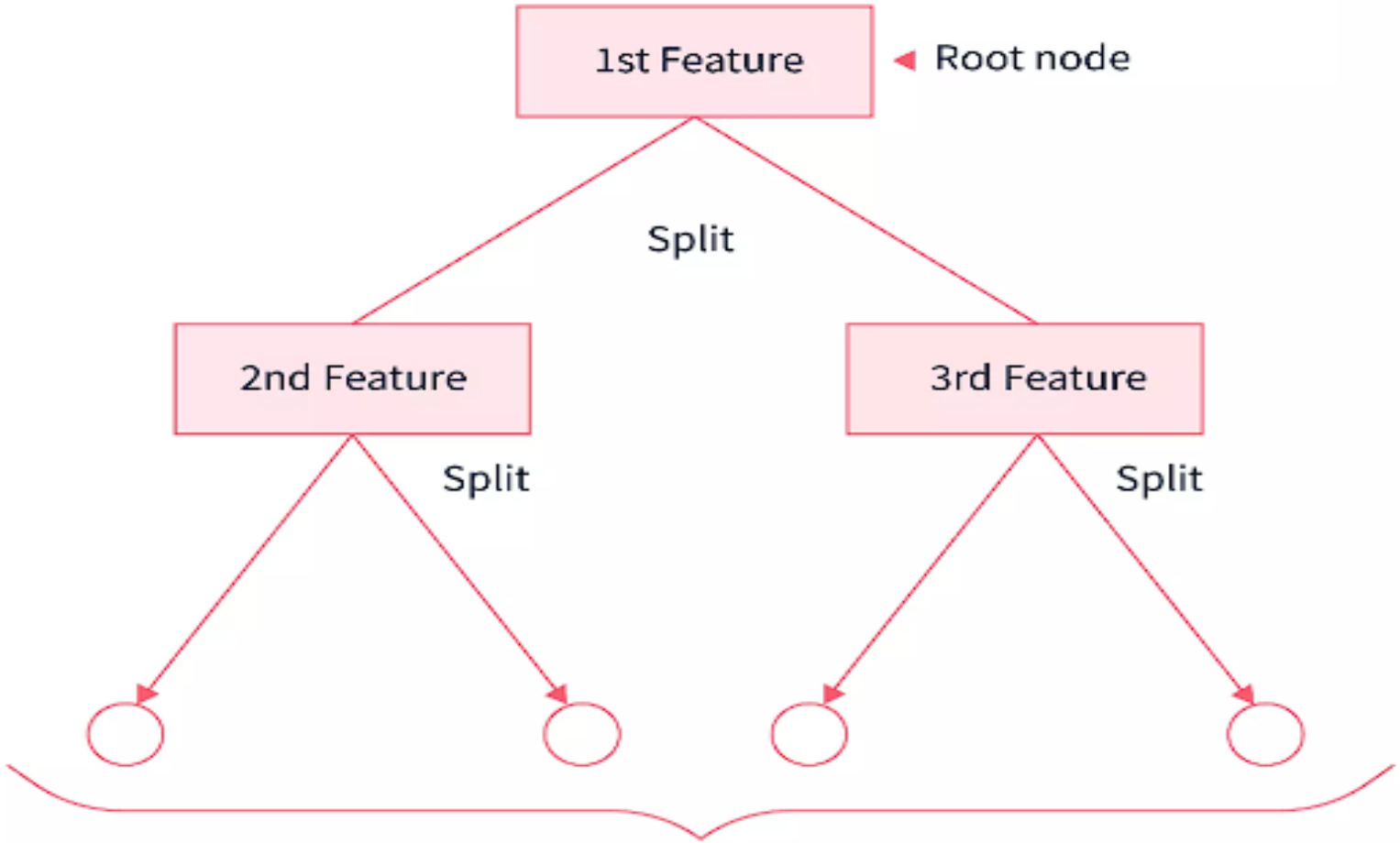

ROOT NODE
College Rankng is Good
Yes
No
DECISION NODE
Placements are good
Not opting for the college
LEAF NODE
Yes
No
DECISION NODE
Distance of college from home
Not opting for the college
LEAF NODE
Yes
No
Opted for the college
LEAF NODE
Not opting for the college
LEAF NODE

Attribute Selection Measures

In general, there are different features/attributes in a dataset. We randomly cannot make a feature/attribute as the root node and then follow the same for every decision node. Doing so would lead to a very wrong model that will make wrong decisions based on unusual features.

So, to solve this problem, we have something called **Attribute Selection Measures** (*ASM*).

Using attribute selection measure, Find best attribute & make it root node



Leaf nodes contain the output of the decisions we make.

π

Here is a list of some attribute selection measures.

1.Gini index

2.Information gain

3.Gain Ratio

4.Entropy

5.Reduction in Variance

*6.Chi-Square***

Let's discuss each one of them:

Entropy :

- It is the measure of impurity (or) uncertainty in the data. It lies between 0 to 1 and is calculated using the below formula.

$$Entropy = \sum_{i=1}^{N} -p_i \times log_2(p_i)$$

- **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
- $-P1log2P1 - P2log2P2 - \ldots\ldots - Pnlog2Pn$

- 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.

Information Gain :

- 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*
- $$Information\ Gain = Entropy(before) - \sum_{i=1}^{N} Entropy(i, after)$$
- 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.

Gini Index :

- It is a measure of purity or impurity while creating a decision tree. It is calculated by subtracting the sum of the squared probabilities of each class from one. It is the same as entropy but is known to calculate quicker as compared to entropy. CART (Classification and regression tree) uses the Gini index as an attribute selection measure to select the best attribute/feature to split. The attribute with a lower Gini index is used as the best attribute to split.

$$Gini = 1 - \sum_{j} p^2 j$$

- Here 'i' is the no. of classes.

Gain Ratio :

- The **Gain Ratio** solves the problem of Bias in Information gain. Information gain shows bias towards the nodes that have many values i.e. if an attribute has a large set of values in the data set, then makes it a root node. This is something that isn't appreciated while designing a model, so to solve this problem of bias, the number of branches that a node would split into is something that is taken under consideration by the gain ratio while splitting a node. The gain ratio is defined as the ratio of information gain and intrinsic information (or) split info.
- Here intrinsic information/split info refers to the entropy of sub-dataset proportions.
- The formula for gain ratio is:

$$\text{Gain Ratio} = \frac{\text{Information Gain}}{\text{Split Info (or) Intrinsic info}}$$

Reduction in variance :

- It is used when the data we have is continuous in nature. Variance simply refers to the change in the model when using different subparts of the training data. The split with lower variance is taken into consideration. To use variance as an attribute selection measure, firstly calculate variance for each node followed by calculating variance for each split.

$$Variance = \frac{\sum(X - \bar{X})^2}{n}$$

- X bar is the mean of the values, X is actual, and n is the number of values.

Chi-Square

- **Chi-Square** is a comparison between observed results and expected results. This statistical method is used in CHAID(Chi-square Automatic Interaction Detector). CHAID in itself is a very old algorithm that is not used much these days. The higher the value of chi-square, higher is the difference between the current node and its parent.
- The formula for chi - square is
-

$$x^2 = \sum \frac{(O - E)^2}{E}$$

Where:

x^2 = Chi Square obtained
$\sum$ = The sum of
O = Observed score
E = Expected score

π

Common problems with Decision trees :

- 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.
- Determining how deeply to grow the decision tree.
- Handling continuous attributes.
- Choosing an appropriate attribute selection measure.
- Handling training data with missing attribute values.

Algorithms :

- **ID3** : ID3 stands for *Iterative Dichotomiser 3* . This algorithm iteratively divides features into two or more groups at each step. Here "Iterative" means continuous/repeated and "dichotomiser" means dividing. ID3 follows a top-down approach i.e., the tree is built from the top and at every step greedy approach is applied. The greedy approach means that at each iteration we select the best feature now to create a node and this node is again split after applying some statistical methods. ID3 generally is not a very ideal algorithm as it overfits when the data is continuous.

- **CART**: CART stands for classification and regression trees. As the name suggests CART can also perform both classification and regression-based tasks. CART uses Gini's impurity index as an attribute selection method while splitting a node into further nodes when it's a classification-based use case and uses sum squared error as an attribute selection measure when the use case is regression-based. While CART uses Gini Index as an ASM(attribute selection measure), C4.5 and ID3 use information gain as an ASM.

Algorithms :

- **CHAID**: CHAID stands for Chi-square Automatic Interaction Detector. It is known to be the oldest of all three algorithms in history and is used very less these days. In CHAID chi-square is the attribute selection measure to split the nodes when it's a classification-based use case and uses F-test as an attribute selection measure when it is a regression-based use case. Higher the chi-square value higher is the preference given to that feature. The major difference between CHAID and CART is, CART splits one node into two nodes whereas CHAID splits one node into 2 or more nodes.
- **MARS**: MARS stands for Multivariate adaptive regression splines. It is an algorithm that was specifically designed to handle regression-based tasks, provided, the data is non-linear.

Algorithms :

- **C4.5**: It is better than the ID3 algorithm as it can handle both discrete and continuous data. In C4.5 splitting is done based on Information gain (attribute selection measure) and the feature with the highest Information gain is made the decision node and is further split. C4.5 handles overfitting by the method of pruning i.e it removes the branches/subpart of the tree that does not hold much importance (or) is redundant. To be specific, C4.5 follows post pruning i.e removing branches after the tree is created.

Classification and Regression Tree algorithm (CART):

To build the Decision Tree, CART (Classification and Regression Tree) algorithm is used. It works by selecting the best split at each node based on metrics like Gini impurity or information Gain. To create a decision tree. Here are the basic steps of the CART algorithm:

1. The root node of the tree is supposed to be the complete training dataset.
2. Determine the impurity of the data based on each feature present in the dataset. Impurity can be measured using metrics like the Gini index or entropy for classification and Mean squared error, Mean Absolute Error, friedman_mse, or Half Poisson deviance for regression.
3. Then selects the feature that results in the highest information gain or impurity reduction when splitting the data.
4. For each possible value of the selected feature, split the dataset into two subsets (left and right), one where the feature takes on that value, and another where it does not. The split should be designed to create subsets that are as pure as possible with respect to the target variable.

Classification and Regression Tree algorithm (CART):

5. Based on the target variable, determine the impurity of each resulting subset.
6. For each subset, repeat steps 2–5 iteratively until a stopping condition is met. For example, the stopping condition could be a maximum tree depth, a minimum number of samples required to make a split or a minimum impurity threshold.
7. Assign the majority class label for classification tasks or the mean value for regression tasks for each terminal node (leaf node) in the tree.

Classification and Regression Tree algorithm (CART):

Let the data available at node m be Q_m and it has n_m samples. and t_m as the threshold for node m. then, The classification and regression tree algorithm for classification can be written as :

$$G(Q_m, t_m) = \frac{n_m^{Left}}{n_m} H(Q_m^{Left}(t_m)) + \frac{n_m^{Right}}{n_m} H(Q_m^{Right}(t_m))$$

MSE is the mean square error.

$$MSE_{Q_m} = \sum_{y \epsilon Q_m} (\bar{y}_m - y)^2$$

$$\text{where, } \bar{y}_m = \frac{1}{n_m} \sum_{y \epsilon Q_m} y$$

Classification and Regression Tree algorithm (CART):

- n_m is the number of instances in the left and right subsets at node m.

› To select the parameter, we can write as:

$$t_m = \underset{t_m}{} MSE(Q_m, t_m)$$

Iterative Dichotomiser 3 (ID3).

- **ID3 in brief**
- 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 now to create a node.
- Most generally ID3 is only used for classification problems with nominal features only.

Iterative Dichotomiser 3 (ID3).

- 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.
- We denote our dataset as **S,** entropy is calculated as:
- $$\textbf{Entropy(S)} = -\sum p_i * \log_2(p_i) \; ; \; i = 1 \text{ to } n$$

Iterative Dichotomiser 3 (ID3).

- were,
  n is the total number of classes in the target column (in our case n = 2 i.e., YES and NO)
  p_i is the **probability of class 'i'** or the ratio of "*number of rows with class i in the target column*" to the "*total number of rows*" in the dataset.
- Information Gain for a feature column **A** is calculated as:
- $$\mathbf{IG(S, A) = Entropy(S) - \sum((|S_v| / |S|) * Entropy(S_v))}$$
- where $\mathbf{S_v}$ is the set of rows in **S** for which the feature column **A** has value **v**, $|\mathbf{S_v}|$ is the number of rows in $\mathbf{S_v}$ and likewise $\mathbf{|S|}$ is the number of rows in **S.**

Iterative Dichotomiser 3 (ID3).

- **ID3 Steps**

1. Calculate the Information Gain of each feature.
2. Considering that all rows don't belong to the same class, split the dataset **S** into subsets using the feature for which the Information Gain is maximum.
3. Make a decision tree node using the feature with the maximum Information gain.
4. If all rows belong to the same class, make the current node as a leaf node with the class as its label.
5. Repeat for the remaining features until we run out of all features, or the decision tree has all leaf nodes.

C4.5 Algorithm :

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

C4.5 Algorithm :

- 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.

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

CHAID (Chi-Square Automatic Interaction Detection)

- Were,
- O_i represents the observed frequency and E_i 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.

MARS (Multivariate Adaptive Regression Splines)

- In MARS, a basis function is described as :

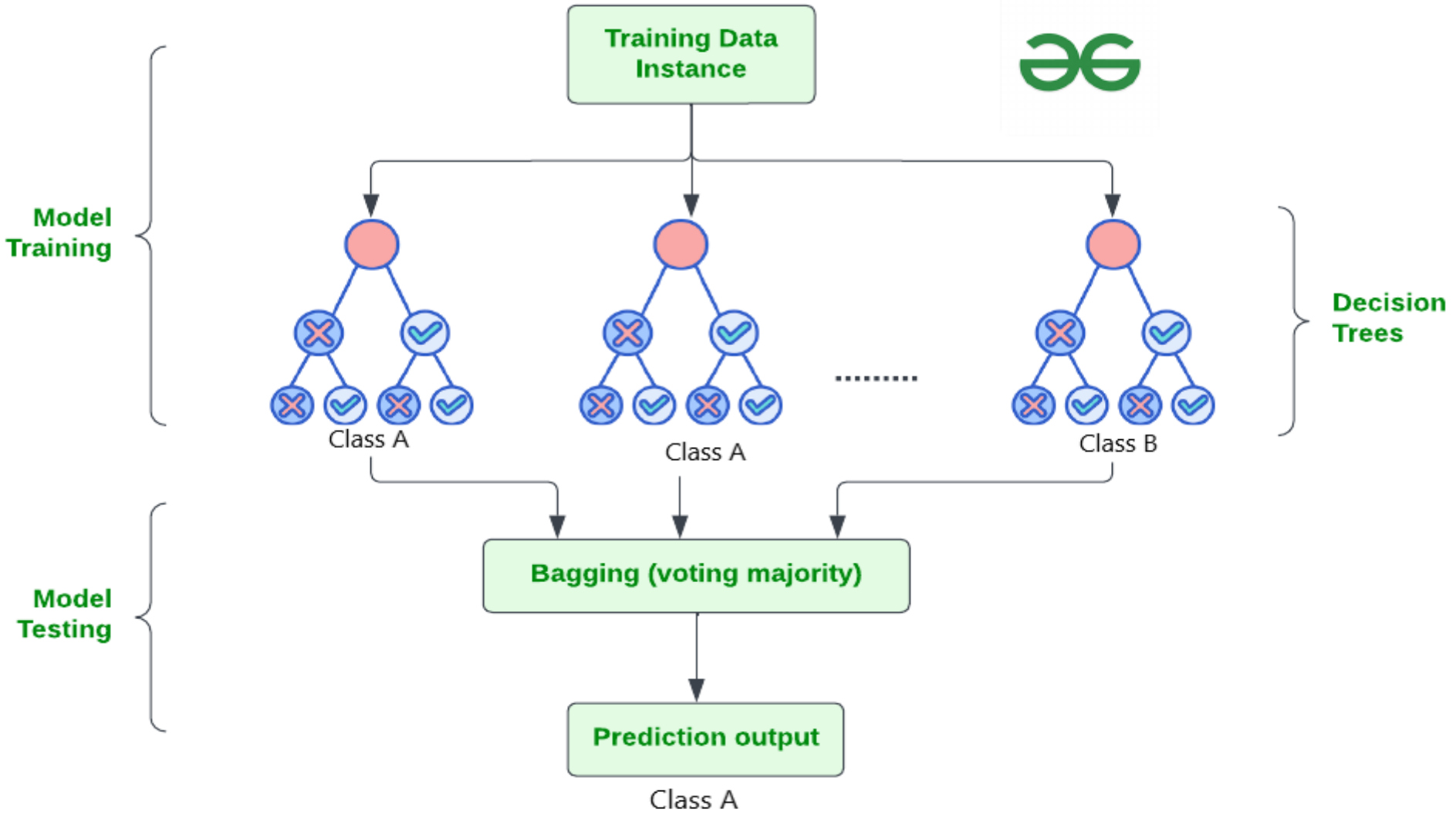
$$h(x) = \begin{cases} x - t \ \ if \ x > t \\ t - x \ \ if x \leq t \end{cases}$$

- 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.

Random Forest Algorithm :

- Random Forest algorithm is a powerful tree learning technique in Machine Learning. It works by creating several Decision Trees during the training phase.
- Each tree is constructed using a random subset of the data set to measure a random subset of features in each partition.
- This randomness introduces variability among individual trees, reducing the risk of overfitting and improving overall prediction performance.
- In prediction, the algorithm aggregates the results of all trees, either by voting (for classification tasks) or by averaging (for regression tasks) This collaborative decision-making process, supported by multiple trees with their insights, provides an example stable and precise results.
- Random forests are widely used for classification and regression functions, which are known for their ability to handle complex data, reduce overfitting, and provide reliable forecasts in different environments.

Random Forest Algorithm :



Random Forest Algorithm :

- The random Forest algorithm works in several steps which are discussed below :
- Ensemble of Decision Trees: Random Forest leverages the power of ensemble learning by constructing an army of Decision Trees. These trees are like individual experts, each specializing in a particular aspect of the data. Importantly, they operate independently, minimizing the risk of the model being overly influenced by the nuances of a single tree.
- Random Feature Selection: To ensure that each decision tree in the ensemble brings a unique perspective, Random Forest employs random feature selection. During the training of each tree, a random subset of features is chosen. This randomness ensures that each tree focuses on different aspects of the data, fostering a diverse set of predictors within the ensemble.
- Bootstrap Aggregating or Bagging: The technique of bagging is a cornerstone of Random Forest's training strategy which involves creating multiple bootstrap samples from the original dataset, allowing instances to be sampled with replacement. This results in different subsets of data for each decision tree, introducing variability in the training process and making the model more robust.
- Decision Making and Voting: When it comes to making predictions, each decision tree in the Random Forest casts its vote. For classification tasks, the final prediction is determined by the mode (most frequent prediction) across all the trees. In regression tasks, the average of the individual tree predictions is taken. This internal voting mechanism ensures a balanced and collective decision-making process.

Random Forest Algorithm :

Some of the Key Features of Random Forest are discussed below :

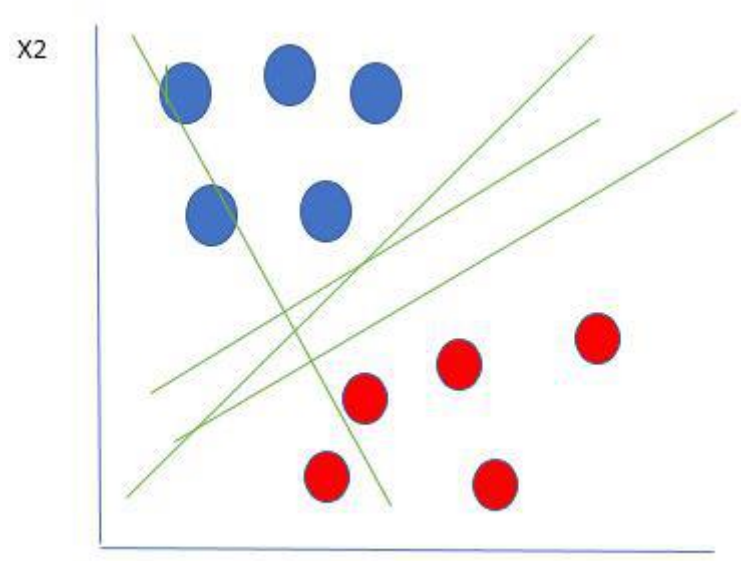
1. **High Predictive Accuracy:** Imagine Random Forest as a team of decision-making wizards. Each wizard (decision tree) looks at a part of the problem, and together, they weave their insights into a powerful prediction tapestry. This teamwork often results in a more accurate model than what a single wizard could achieve.
2. **Resistance to Overfitting:** Random Forest is like a cool-headed mentor guiding its apprentices (decision trees). Instead of letting each apprentice memorize every detail of their training, it encourages a more well-rounded understanding. This approach helps prevent getting too caught up with the training data which makes the model less prone to overfitting.
3. **Large Datasets Handling:** Dealing with a mountain of data? Random Forest tackles it like a seasoned explorer with a team of helpers (decision trees). Each helper takes on a part of the dataset, ensuring that the expedition is not only thorough but also surprisingly quick.
4. **Variable Importance Assessment:** Think of Random Forest as a detective at a crime scene, figuring out which clues (features) matter the most. It assesses the importance of each clue in solving the case, helping you focus on the key elements that drive predictions.
5. **Built-in Cross-Validation:** Random Forest is like having a personal coach that keeps you in check. As it trains each decision tree, it also sets aside a secret group of cases (out-of-bag) for testing. This built-in validation ensures your model doesn't just ace the training but also performs well on new challenges.

SVM-Support Vector Machine:

INTRODUCTION :

- Support Vector Machine :
- Support Vector Machine (SVM) is a supervised machine learning algorithm used for both classification and regression. Though we say regression problems as well it's best suited for classification. The main objective of the SVM algorithm is to find the optimal hyperplane in an N-dimensional space that can separate the data points in different classes in the feature space. The hyperplane tries that the margin between the closest points of different classes should be as maximum as possible. The dimension of the hyperplane depends upon the number of features. If the number of input features is two, then the hyperplane is just a line. If the number of input features is three, then the hyperplane becomes a 2-D plane. It becomes difficult to imagine when the number of features exceeds three.

- Let's consider two independent variables x1, x2, and one dependent variable which is either a blue circle or a red circle.

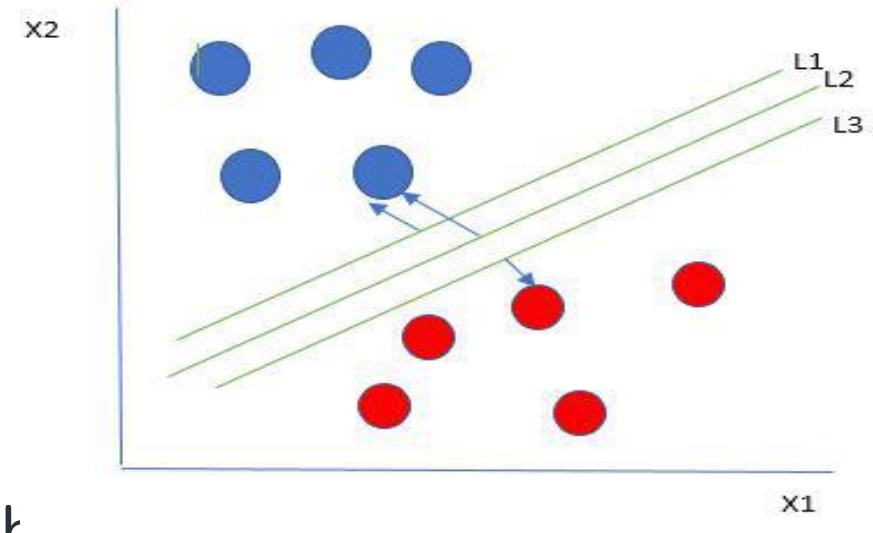


Linearly Separable Data points

From the figure above it's very clear that there are multiple lines (our hyperplane here is a line because we are considering only two input features x_1, x_2) that segregate our data points or do a classification between red and blue circles. So how do we choose the best line or in general the best hyperplane that segregates our data points?

How does SVM work?

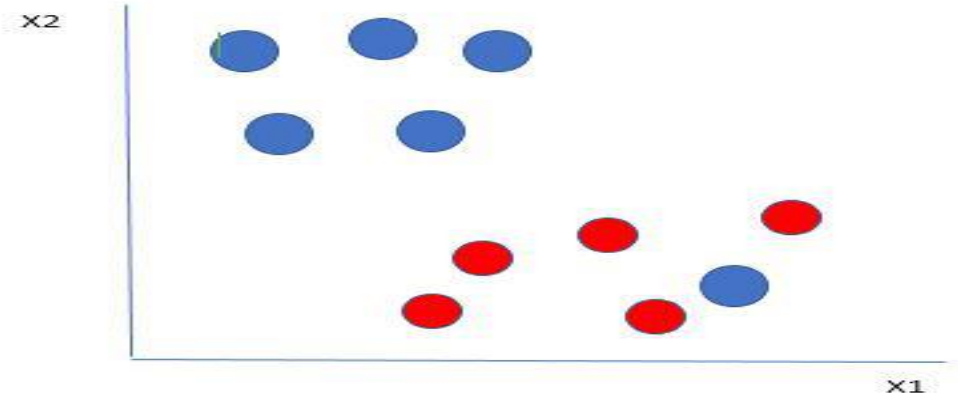
One reasonable choice as the best hyperplane is the one that represents the largest separation or margin between the two classes.



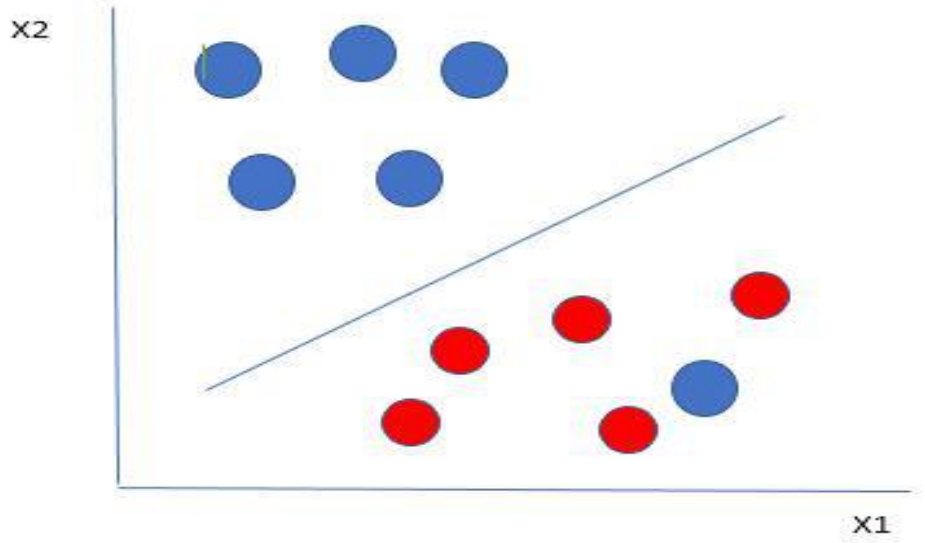
So, we choose the hyperplane whose distance from it to the nearest data point on each side is maximized. If such a hyperplane exists it is known as the **maximum-margin hyperplane/hard margin**. So, from the above figure, we choose L2. Let's consider a scenario like shown below

How does SVM work?

›



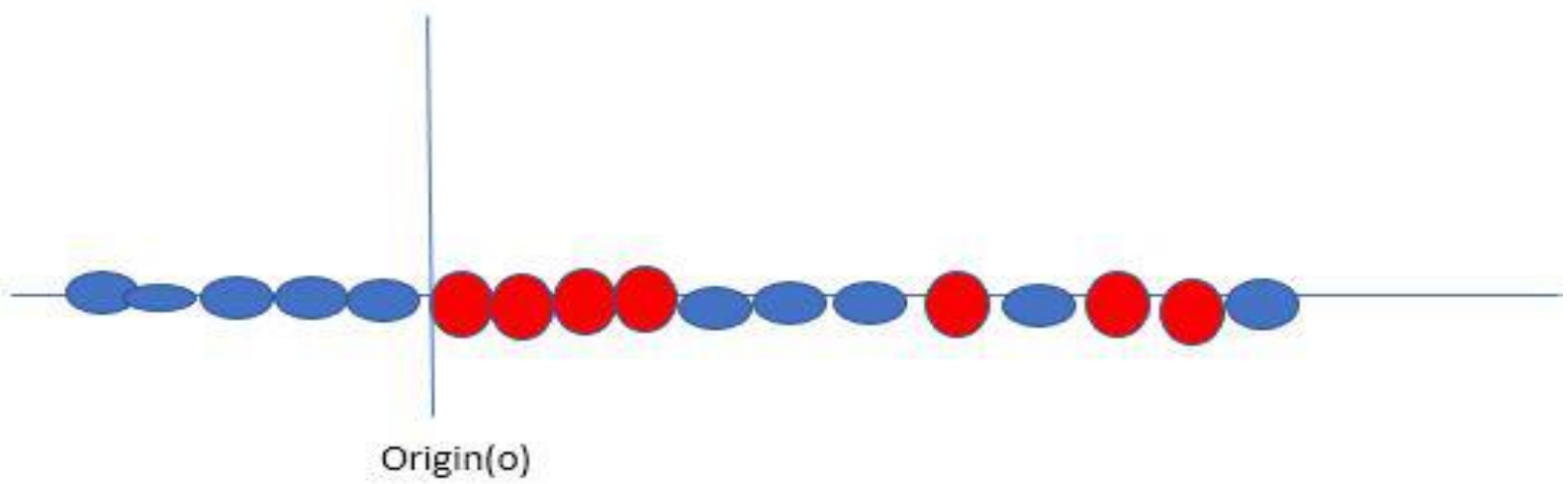
Here we have one blue ball in the boundary of the red ball. So how does SVM classify the data? It's simple! The blue ball in the boundary of red ones is an outlier of blue balls. The SVM algorithm has the characteristics to ignore the outlier and finds the best hyperplane that maximizes the margin. SVM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



How does SVM work?

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+Λ(∑penalty)). Hinge loss is a commonly used penalty. If no violations no hinge loss. If violations hinge loss proportional to the distance of violation.

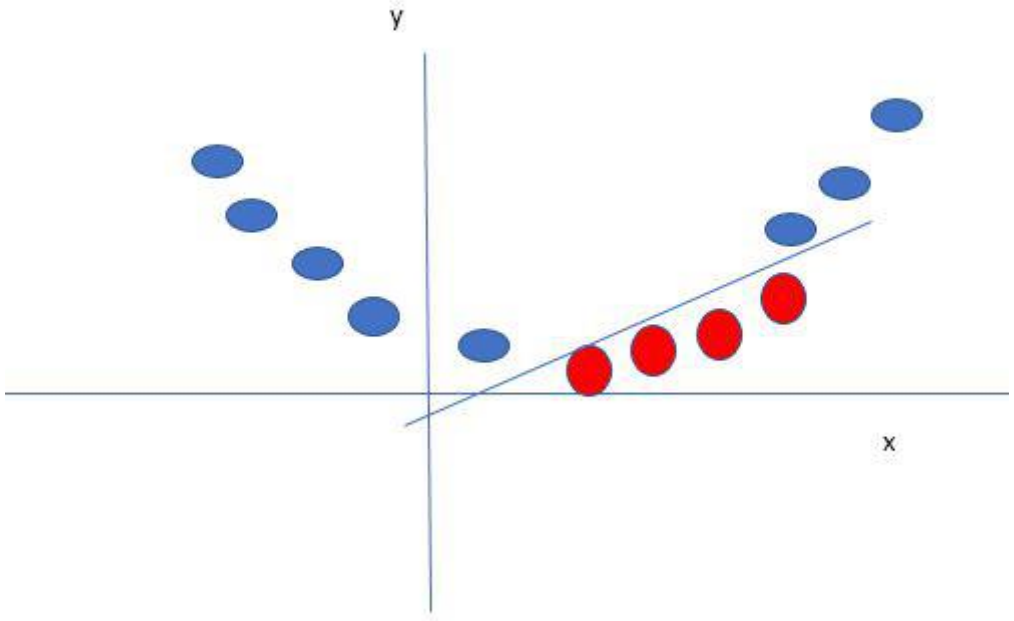
Till now, we were talking about linearly separable data(the group of blue balls and red balls are separable by a straight line/linear line). What to do if data are not linearly separable?



How does SVM work?

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.

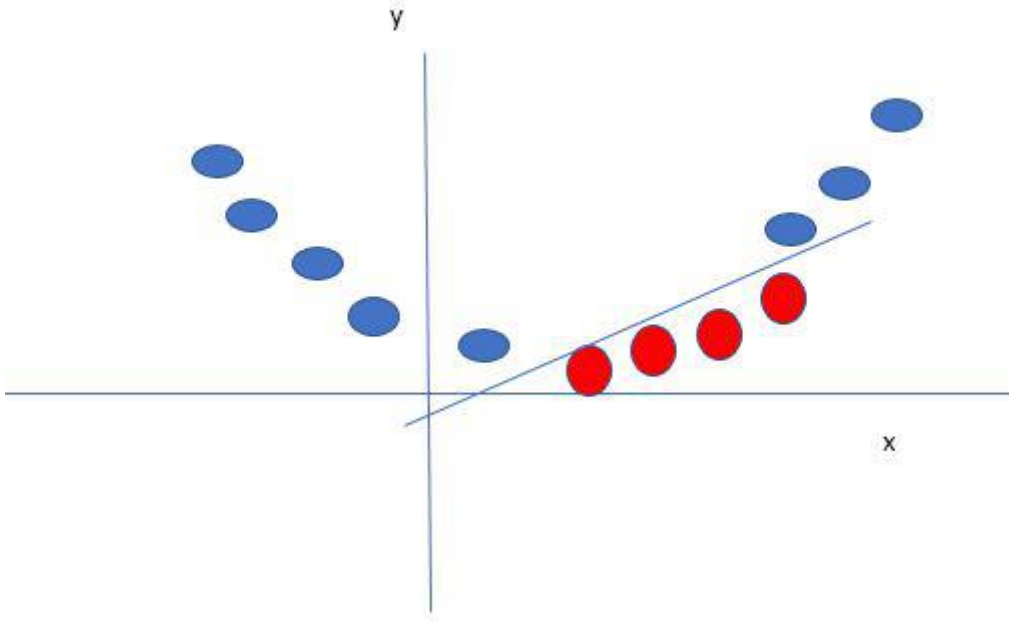


Mapping 1D data to 2D to become able to separate the two classes

How does SVM work?

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