UNIT IV: Supervised Learning II: Decision Trees: ID3, Classification and Regression Trees, Regression: Linear Regression, Multiple Linear Regression and Logistic Regression.

Neural Networks: Introduction, Perceptron, Multilayer Perceptron, Support vector machines: Linear and Non-Linear, Kernel Functions, K-Nearest Neighbors.

QUESTION BANK

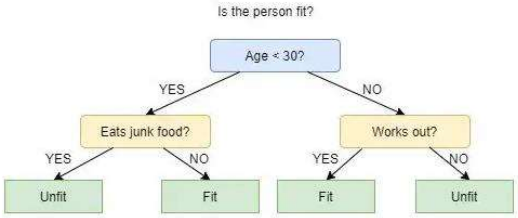
1. Discuss in detail about decision tree ID3.
2. Explain Linear Regression with its cost function.
3. Briefly explain about Multiple Linear Regression with an example.
4. Explain Logistic Regression with an example
5. Differentiate Classification and Regression with an example.
6. Explain Support vector machines and its types.
7. Explain K-Nearest Neighbors and its working in detail
8. Explain working of Neural Network.
9. Discuss briefly about CART in decision trees.
10. Explain multi-layer perceptron architecture.

1. Discuss in detail about decision tree ID3.

Decision Tress: ID3

- In simple words, a decision tree is a structure that contains nodes (rectangular boxes) and edges(arrows) and is built from a dataset (table of columns representing features/attributes and rows corresponds to records).
- Each node is either used to ***make a decision*** *(*known as decision node) or ***represent an outcome*** (known as leaf node).

Decision tree Example



The picture above depicts a decision tree that is used to classify whether a person is **Fit** or **Unfit.**

The decision nodes here are questions like "*'Is the person less than 30 years of age?'*, *'Does the person eat junk?'*, etc. and the leaves are one of the two possibl outcomes viz. **Fit** and **Unfit**.
Looking at the Decision Tree we can say make the following decisions:
if a person is less than 30 years of age and doesn't eat junk food then he is Fit, if a person is less than 30 years of age and eats junk food then he is Unfit and so on.

The initial node is called the **root node** *(colored in blue)*, the final nodes are called the **leaf nodes** *(colored in green)* and the rest of the nodes are called **intermediate** or **internal** nodes.
The root and intermediate nodes represent the decisions while the leaf node represent the outcomes.

ID3 in brief

ID3 stands for Iterative Dichotomiser 3 and is named such because the algorithm iteratively (repeatedly) dichotomizes(divides) features into two more groups at each step.

Invented by Ross Quinlan, ID3 uses a **top-down greedy** approach to build decision tree. In simple words, the **top-down** approach means that we sta building the tree from the top and the **greedy** approach means that at eac iteration we select the best feature at the present moment to create a nod

Most generally ID3 is only used for classification problems with nominal features only.

Dataset description
we'll be using a sample dataset of COVID-19 infection.

ID	Fever	Cough	Breathing issues	Infected
1	NO	NO	NO	NO
2	YES	YES	YES	YES
3	YES	YES	NO	NO
4	YES	NO	YES	YES
5	YES	YES	YES	YES
6	NO	YES	NO	NO
7	YES	NO	YES	YES
8	YES	NO	YES	YES
9	NO	YES	YES	YES
10	YES	YES	NO	YES
11	NO	YES	NO	NO
12	NO	YES	YES	YES
13	NO	YES	YES	NO
14	YES	YES	NO	NO

Metrics in ID3

As mentioned previously, the ID3 algorithm selects the best feature at each step while building a Decision tree.

Before you ask, the answer to the question: 'How does ID3 select the best feature?' is that ID3 uses **Information Gain** or just **Gain** to find the best feature.

Information Gain calculates the reduction in the entropy and measures how well a given feature separates or classifies the target classes. The feature with the **highest Information Gain** is selected as the **best** one.

In simple words, **Entropy** is the measure of disorder and the Entropy of a dataset is the measure of disorder in the target feature of the dataset.
In the case of binary classification (where the target column has only two types of classes) entropy is **0** if all values in the target column are homogenous(similar) and will be **1** if the target column has equal number values for both the classes.

We denote our dataset as **S**, entropy is calculated as:

$$Entropy(S) = - \sum p_i * \log_2(p_i) \; ; \; i = 1 \text{ to } n$$

where,
n is the total number of classes in the target column (in our case n = 2 i.e YES and NO)
$\mathbf{p_i}$ 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:

$$IG(S, A) = Entropy(S) - \sum((|S_v| \; / \; |S|) * Entropy(S_v))$$

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 **|S|** is the number of rows in **S**.

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.

Implementation on our Dataset

As stated in the previous section the first step is to find the best feature i.e. the one that has the maximum Information Gain(**IG**). We'll calculate the IG for each of the features now, but for that, we first need to calculate the entropy of **S**

From the total of 14 rows in our dataset **S**, there are **8** rows with the target value **YES** and **6** rows with the target value **NO**. The entropy of **S** is calculated as:

$$Entropy(S) = -(8/14) * \log_2(8/14) - (6/14) * \log_2(6/14) = 0.99$$

We now calculate the Information Gain for each feature:

IG calculation for Fever:

In this(Fever) feature there are **8** rows having value **YES** and **6** rows having value **NO.**

As shown below, in the **8** rows with **YES for** Fever, there are **6** rows having target value **YES** and **2** rows having target value **NO.**

Fever	Cough	Breathing issues	Infected
YES	YES	YES	YES
YES	YES	NO	NO
YES	NO	YES	YES
YES	YES	YES	YES
YES	NO	YES	YES
YES	NO	YES	YES
YES	YES	NO	YES
YES	YES	NO	NO

As shown below, in the **6** rows with **NO**, there are **2** rows having target value **YES** and **4** rows having target value **NO.**

Fever	Cough	Breathing issues	Infected
NO	NO	NO	NO
NO	YES	NO	NO
NO	YES	YES	YES
NO	YES	NO	NO
NO	YES	YES	YES
NO	YES	YES	NO

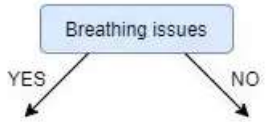
The block, below, demonstrates the calculation of Information Gain for **Fever.**

```
# total rows
|S| = 14

For v = YES, |Sᵥ| = 8
Entropy(Sᵥ) = - (6/8) * log₂(6/8) - (2/8) * log₂(2/8) = 0.81

For v = NO, |Sᵥ| = 6
Entropy(Sᵥ) = - (2/6) * log₂(2/6) - (4/6) * log₂(4/6) = 0.91

# Expanding the summation in the IG formula:
IG(S, Fever) = Entropy(S) - (|SYES| / |S|) * Entropy(SYES) -
(|SNO| / |S|) * Entropy(SNO)

∴ IG(S, Fever) = 0.99 - (8/14) * 0.81 - (6/14) * 0.91 = 0.13
```

Next, we calculate the **IG** for the features **"Cough"** and **"Breathing issues".**

```
IG(S, Cough) = 0.04
IG(S, BreathingIssues) = 0.40
```

Since the feature **Breathing issues** have the highest Information Gain it is used to create the root node.
Hence, after this initial step our tree looks like this:



Next, from the remaining two unused features, namely, **Fever** and **Cough**, we decide which one is the best for the left branch of **Breathing Issues**.
Since the left branch of **Breathing Issues** denotes **YES,** we will work with the subset of the original data i.e the set of rows having **YES** as the value in the Breathing Issues column. These **8 rows** are shown below:

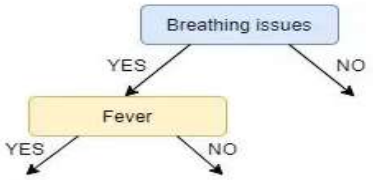
Fever	Cough	Breathing issues	Infected
YES	YES	YES	YES
YES	NO	YES	YES
YES	YES	YES	YES
YES	NO	YES	YES
YES	NO	YES	YES
NO	YES	YES	YES
NO	YES	YES	YES
NO	YES	YES	NO

Next, we calculate the IG for the features Fever and Cough using the subset S_{BY} (Set **Breathing** Issues **Y**es) which is shown above :

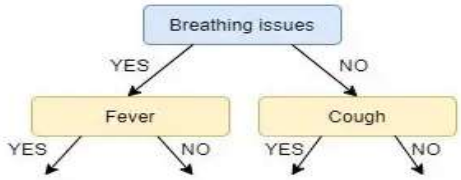
*Note: For **IG** calculation the Entropy will be calculated from the subset S_{BY} and not the original dataset **S**.*

$IG(S_{BY}, Fever) = 0.20$
$IG(S_{BY}, Cough) = 0.09$

IG of Fever is greater than that of Cough, so we select **Fever** as the left branch of Breathing Issues:
Our tree now looks like this:



Next, we find the feature with the maximum IG for the right branch of **Breathing Issues**. But, since there is only one unused feature left we have no other choice but to make it the right branch of the root node.
So our tree now looks like this:



There are no more unused features, so we stop here and jump to the final step of creating the leaf nodes.
For the left leaf node of Fever, we see the subset of rows from the original data set that has **Breathing Issues** and **Fever** both values as **YES**.

Fever	Cough	Breathing issues	Infected
YES	YES	YES	YES
YES	NO	YES	YES
YES	YES	YES	YES
YES	NO	YES	YES
YES	NO	YES	YES

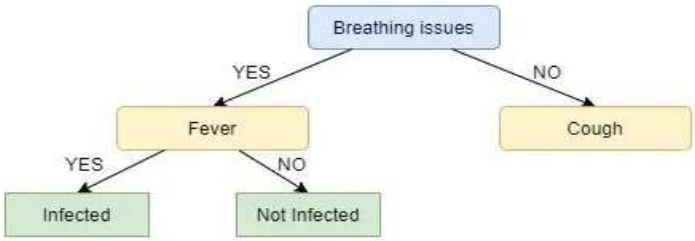
Since all the values in the target column are **YES,** we label the left leaf node as **YES**, but to make it more logical we label it **Infected.**

Similarly, for the right node of Fever we see the subset of rows from the original data set that have **Breathing Issues** value as **YES** and **Fever** as **NO**.

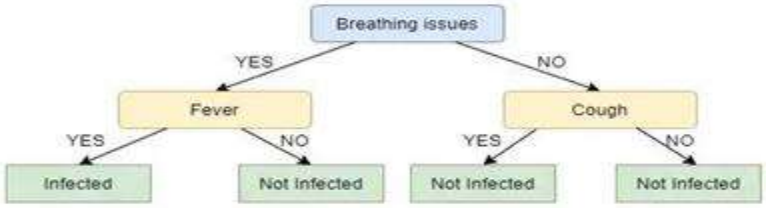
Fever	Cough	Breathing issues	Infected
NO	YES	YES	YES
NO	YES	YES	NO
NO	YES	YES	NO

Here not all but **most** of the **values** are **NO,** hence **NO** or **Not Infected** becomes our **right leaf node.**

Our tree, now, looks like this:



We repeat the same process for the node **Cough**, however here both left and right leaves turn out to be the same i.e. **NO** or **Not Infected** as shown below:



The right node of Breathing issues is as good as just a leaf node with class 'Not infected'. This is one of the Drawbacks of ID3, it doesn't do pruning.

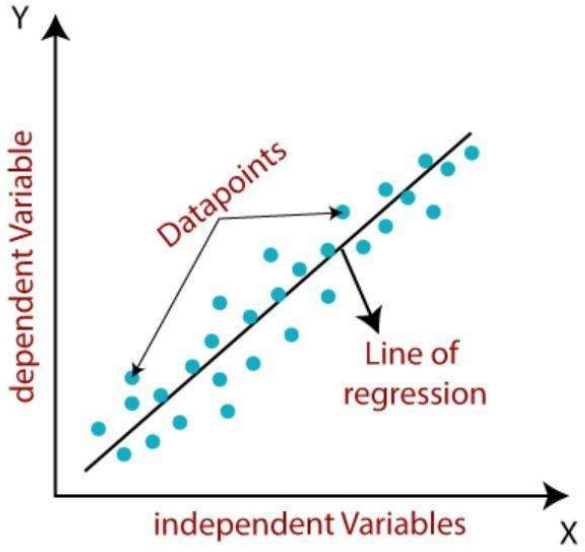
Pruning is a mechanism that reduces the size and complexity of a Decision tree by removing unnecessary nodes.

Another drawback of ID3 is overfitting or high variance i.e. it learns the dataset it used so well that it fails to generalize on new data.

2.Explain Linear Regression with its cost function.

Linear Regression

- Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis.
- Linear regression makes predictions for continuous/real or numeric variables such as **sales, salary, age, product price,** etc.
- Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression.
- Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.
- The linear regression model provides a sloped straight line representing the relationship between the variables. Consider the below image:



Mathematically, we can represent a linear regression as:

$$y= a_0+a_1x+ \varepsilon$$

Here,

- Y= Dependent Variable (Target Variable)
- X= Independent Variable (predictor Variable)
- a0= intercept of the line (Gives an additional degree of freedom)
- a1 = Linear regression coefficient (scale factor to each input value).

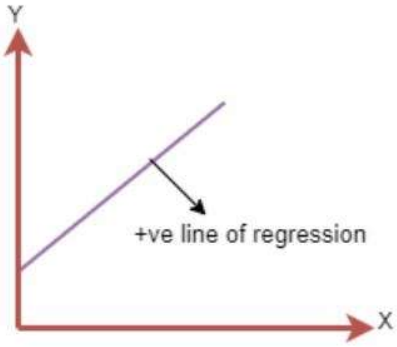
- ε = random error
- The values for x and y variables are training datasets for Linear Regression model representation.

Linear Regression Line

- A linear line showing the relationship between the dependent and independent variables is called a **regression line**. A regression line can show two types of relationship:

Positive Linear Relationship:

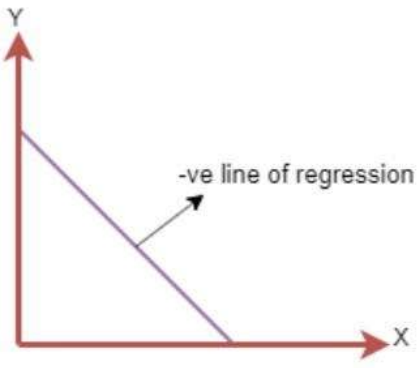
- If the dependent variable increases on the Y-axis and independent variable increases on X-axis, then such a relationship is termed as a Positive linear relationship.



The line equation will be: $Y= a_0+a_1x$

Negative Linear Relationship:

- If the dependent variable decreases on the Y-axis and independent variable increases on the X-axis, then such a relationship is called a negative linear relationship.



The line of equation will be: $Y= -a_0+a_1x$

Finding the best fit line:

- When working with linear regression, our main goal is to find the best fit line that means the error between predicted values and actual values should be minimized. The best fit line will have the least error.

- The different values for weights or the coefficient of lines (a_0, a_1) gives a different line of regression, so we need to calculate the best values for a_0 and a_1 to find the best fit line, so to calculate this we use cost function.

Cost function-

- The different values for weights or coefficient of lines (a_0, a_1) gives the different line of regression, and the cost function is used to estimate the values of the coefficient for the best fit line.

- Cost function optimizes the regression coefficients or weights. It measures how a linear regression model is performing.

- We can use the cost function to find the accuracy of the **mapping function**, which maps the input variable to the output variable. This mapping function is also known as **Hypothesis function**.

- We use the **Mean Squared Error (MSE)** cost function, which is the average of squared error occurred between the predicted values and actual values.

For the above linear equation, MSE can be calculated as:

$$MSE = 1\frac{1}{N}\sum_{i=1}^{n}(y_i - (a_1x_i + a_0))^2$$

Where,

N=Total number of observation

Yi = Actual value

($a1x_i+a_0$)= Predicted value.

3.Briefly explain about Multiple Linear Regression with an example.

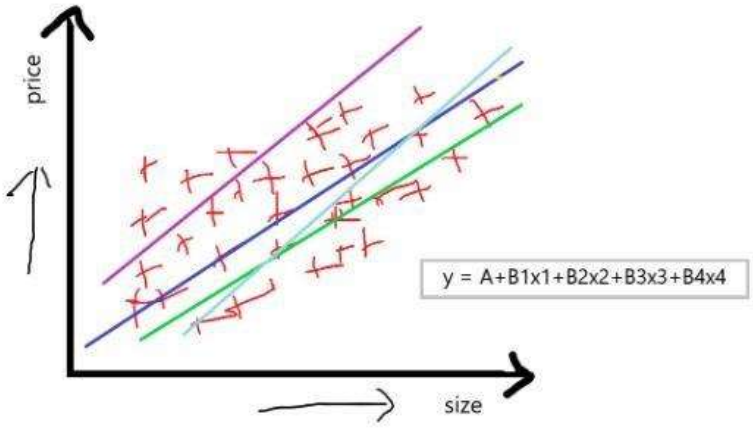
Multiple Linear Regression

- Multiple Linear Regression is basically indicating that we will be having many features Such as f1, f2, f3, f4, and our output feature f5. If we take the same example as above we discussed, suppose:

- f1 is the size of the house.
- f2 is bad rooms in the house.
- f3 is the locality of the house.
- f4 is the condition of the house and,
- f5 is our output feature which is the price of the house.
- Now, you can see that multiple independent features also make a huge impact on the price of the house, price can vary from feature to feature. When we are discussing multiple linear regression then the equation of simple linear regression y=A+Bx is converted to something like:

equation: y = A+B1x1+B2x2+B3x3+B4x4

"If we have one dependent feature and multiple independent features then basically call it a multiple linear regression."



- Now, our aim to using the multiple linear regression is that we have to compute A which is an intercept, and B1 B2 B3 B4 which are the slops or coefficient concerning this independent feature, that basically indicates that if we increase the value of x1 by 1 unit

then B1 says that how much value it will affect int he price of the house, and this was similar concerning others B2 B3 B4

- So, this is a small theoretical description of multiple linear regression now we will use the scikit learn linear regression library to solve the multiple linear regression problem.

Dataset

- Now, we apply multiple linear regression on the 50_startups dataset, you can click here to download the dataset.

Reading dataset

- Most of the dataset are in CSV file, for reading this file we use pandas library:

```
df = pd.read_csv('50_Startups.csv')
df
```

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

Here you can see that there are 5 columns in the dataset where the state stores the categorical data points, and the rest are numerical features.

Now, we have to classify independent and dependent features:

Independent and Dependent variables

There are total 5 features in the dataset, in which basically profit is our dependent feature, and the rest of them are our independent features:

Handling categorical variables

In our dataset, there is one categorical column State, we have to handle this categorical values present inside this column for that we will use pandas get_dummies() function:

 # handle categorical variable

states=pd.get_dummies(x,drop_first=True)

 # dropping extra column

x= x.drop('State',axis=1)

 # concatation of independent variables and new cateorical variable.

x=pd.concat([x,states],axis=1) x

	R&D Spend	Administration	Marketing Spend	R&D Spend	Administration	Marketing Spend	State_Florida	State_New York
0	165349.20	136897.80	471784.10	165349.20	136897.80	471784.10	0	1
1	162597.70	151377.59	443898.53	162597.70	151377.59	443898.53	0	0
2	153441.51	101145.55	407934.54	153441.51	101145.55	407934.54	1	0
3	144372.41	118671.85	383199.62	144372.41	118671.85	383199.62	0	1
4	142107.34	91391.77	366168.42	142107.34	91391.77	366168.42	1	0
5	131876.90	99814.71	362861.36	131876.90	99814.71	362861.36	0	1
6	134615.46	147198.87	127716.82	134615.46	147198.87	127716.82	0	0
7	130298.13	145530.06	323876.68	130298.13	145530.06	323876.68	1	0
8	120542.52	148718.95	311613.29	120542.52	148718.95	311613.29	0	1

Splitting Data

Now, we have to split the data into training and testing parts for that we use the scikitlearn train_test_split() function.

```
# importing train_test_split from sklearn
from sklearn.model_selection import train_test_split
# splitting the data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 42)
```

Applying model

Now, we apply the linear regression model to our training data, first of all, we have to import linear regression from the scikit-learn library, there is no other library to implement multiple linear regression we do it with linear regression only.

```
# importing module
from sklearn.linear_model import LinearRegression
# creating an object of LinearRegression class
LR = LinearRegression()
# fitting the training data
LR.fit(x_train,y_train)
```

Finally, if we execute this then our model will be ready, now we have x_test data we use this data for the prediction of profit.

```
y_prediction =  LR.predict(x_test)
y_prediction
```

```
array([126362.87908253,  84608.45383639,  99677.49425151,  46357.46068582,
       128750.48288501,  50912.41741892, 109741.35032701, 100643.24281646,
        97599.27574597, 113097.42524434])
```

Now, we have to compare the y_prediction values with the original values because we have to calculate the accuracy of our model, which was implemented by a concept called r2_score.

r2_score:-

It is a function inside sklearn.metrics module, where the value of r2_score varies between 0 and 100 percent; we can say that it is closely related to MSE.

r2 is basically calculated by the formula given below:

formula: r2 = 1 – (SS_{res} /SS_{mean})

now, when I say SS_{res} it means, it is the sum of residuals and SS_{mean} refers to the sum of means. where,

$$SS_{res} = \frac{1}{n}\sum_{i=1}^{n}(y - \hat{y})^2$$

y = original values **y^** =

predicted values. and,

$$SS_{mean} = \frac{1}{n}\sum_{i=1}^{n}(y - \bar{y}$$

4.Explain Logistic Regression with an example

Logistic Regression

- Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.

- Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, **it gives the probabilistic values which lie between 0 and 1**.

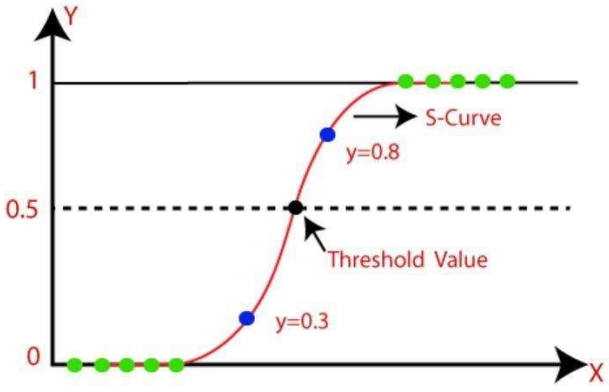
- Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas **Logistic regression is used for solving the classification problems**.

- In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).

- The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.

- Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.

- Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification. The below image is showing the logistic function:



Logistic Function (Sigmoid Function):

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.

- It maps any real value into another value within a range of 0 and 1.

- The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the sigmoid function or the logistic function.

- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

- **Linear Regression** is a machine learning algorithm based on **supervised regression algorithm**. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting.

- **Logistic regression** is basically a **supervised classification algorithm**. In a classification problem, the target variable (or output), y, can take only discrete values for a given set of features (or inputs), X.

5.Differentiate Classification and Regression with an example.

Classification and Regression are two major prediction problems that are usually dealt with in Data Mining and Machine Learning. We are going to deal with both Classification and Regression and we will also see differences between them in this article.

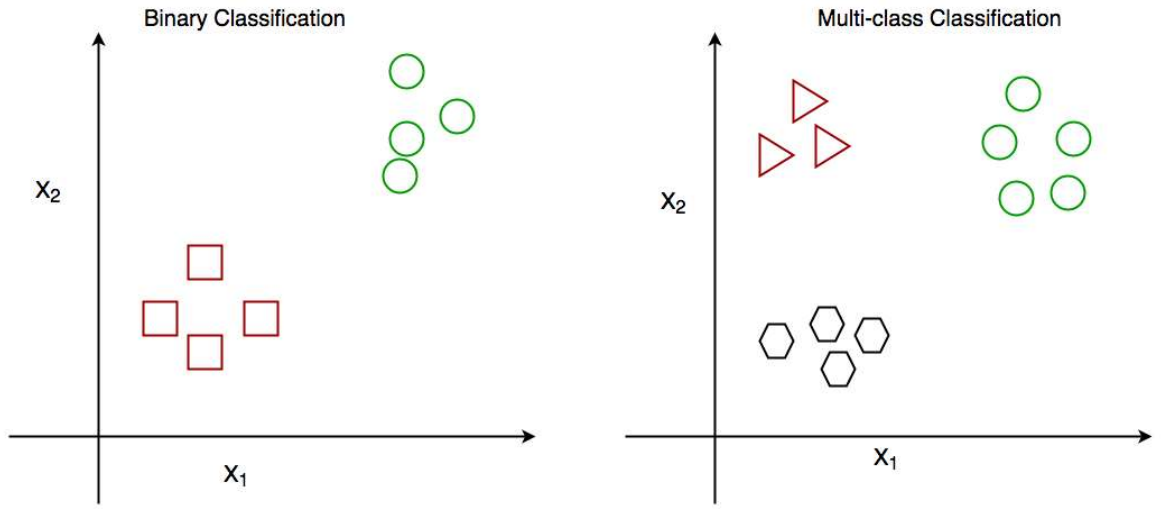
Classification Algorithms

Classification is the process of finding or discovering a model or function that helps in separating the data into multiple categorical classes i.e. discrete values. In classification, data is categorized under different labels according to some parameters given in the input and then the labels are predicted for the data.

- In a classification task, we are supposed to predict discrete target variables(class labels) using independent features.

- In the classification task, we are supposed to find a decision boundary that can separate the different classes in the target variable.

The derived mapping function could be demonstrated in the form of “IF-THEN” rules. The classification process deals with problems where the data can be divided into binary or multiple discrete labels. Let’s take an example, suppose we want to predict the possibility of the winning of a match by Team A on the basis of some parameters recorded earlier. Then there would be two labels Yes and No.



Binary Classification and Multiclass Classification

Types of Classification Algorithms

There are different types of State of the art classification algorithms that have been developed over time to give the best results for classification tasks by employing techniques like bagging and boosting.

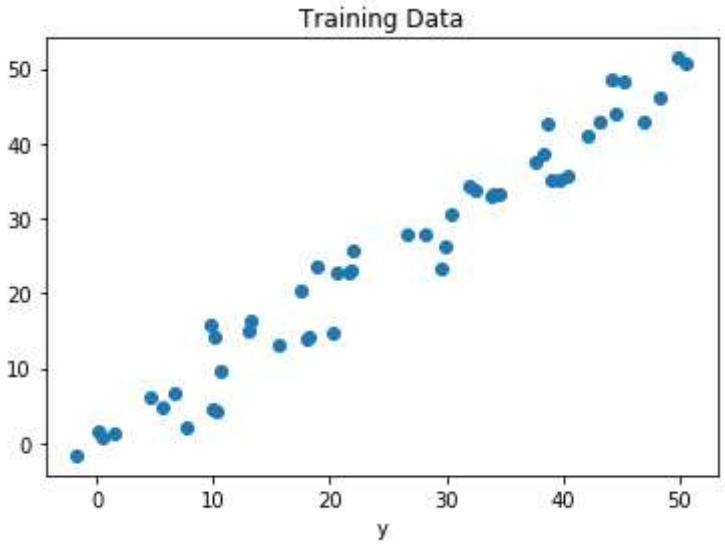
- Decision Tree
- Random Forest Classifier
- K – Nearest Neighbors
- Support Vector Machine

Regression Algorithms

Regression is the process of finding a model or function for distinguishing the data into continuous real values instead of using classes or discrete values. It can also identify the distribution movement depending on the historical data. Because a regression predictive model predicts a quantity, therefore, the skill of the model must be reported as an error in those predictions.

- In a regression task, we are supposed to predict a continuous target variable using independent features.
- In the regression tasks, we are faced with generally two types of problems linear and non-linear regression.

Let's take a similar example in regression also, where we are finding the possibility of rain in some particular regions with the help of some parameters recorded earlier. Then there is a probability associated with the rain.



Regression of Day vs Rainfall (in mm)

Types of Regression Algorithms

There are different types of State of the art regression algorithms that have been developed over time to give the best results for regression tasks by employing techniques like bagging and boosting.

- Lasso Regression
- Ridge Regression
- XGBoost Regressor
- LGBM Regressor

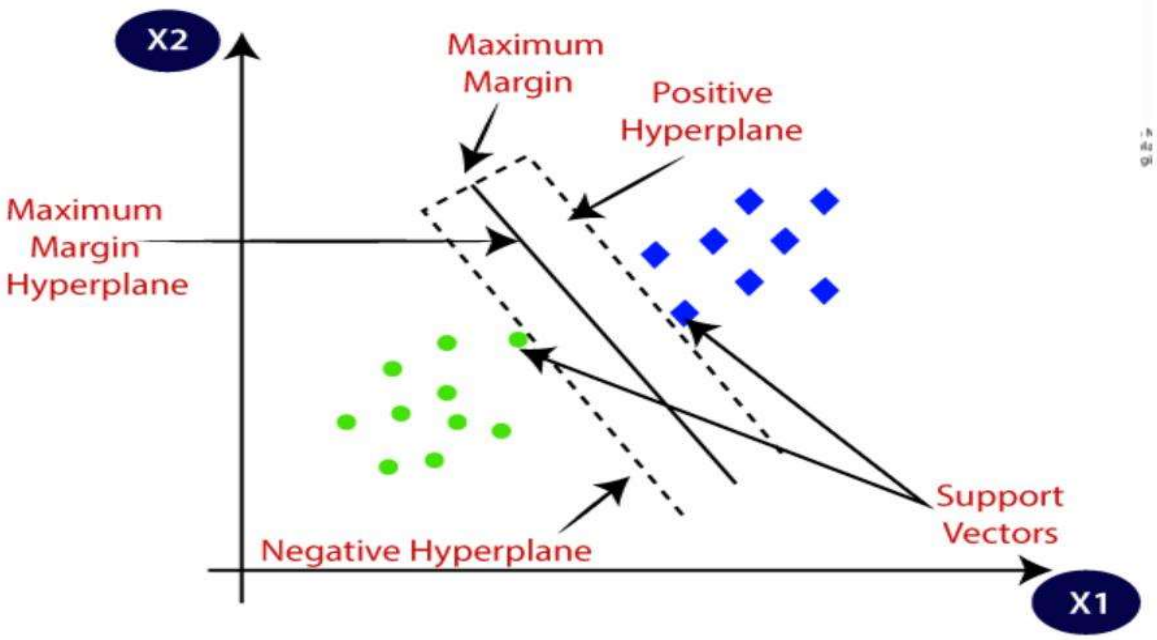
Comparison between Classification and Regression

Classification	Regression
In this problem statement, the target variables are discrete.	In this problem statement, the target variables are continuous.
Problems like Spam Email Classification, Disease prediction like problems are solved using Classification Algorithms.	Problems like House Price Prediction, Rainfall Prediction like problems are solved using regression Algorithms.
In this algorithm, we try to find the best possible decision boundary which can separate the two classes with the maximum possible separation.	In this algorithm, we try to find the best-fit line which can represent the overall trend in the data.

Classification	Regression
Evaluation metrics like Precision, Recall, and F1-Score are used here to evaluate the performance of the classification algorithms.	Evaluation metrics like Mean Squared Error, R2-Score, and MAPE are used here to evaluate the performance of the regression algorithms.
Here we face the problems like binary Classification or Multi-Class Classification problems.	Here we face the problems like Linear Regression models as well as non-linear models.
Input Data are Independent variables and categorical dependent variable.	Input Data are Independent variables and continuous dependent variable.
The classification algorithm's task mapping the input value of x with the discrete output variable of y.	The regression algorithm's task is mapping input value (x) with continuous output variable (y).
Output is Categorical labels.	Output is Continuous numerical values.
Objective is to Predict categorical/class labels.	Objective is to Predicting continuous numerical values.
Example use cases are Spam detection, image recognition, sentiment analysis	Example use cases are Stock price prediction, house price prediction, demand forecasting.
Examples of classification algorithms are: Logistic Regression, Decision Trees, Random Forest, Support Vector Machines (SVM), K-Nearest Neighbors (K-NN), Naive Bayes, Neural Networks, K-Means Clustering, Multi-layer Perceptron (MLP), etc.	**Examples of regression algorithms are:** Linear Regression, Polynomial Regression, Ridge Regression, Lasso Regression, Support Vector Regression (SVR), Decision Trees for Regression, Random Forest Regression, K-Nearest Neighbors (K-NN) Regression, Neural Networks for Regression, etc.

6.Explain Support vector machines and its types.

- 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
- 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:



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.

  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:

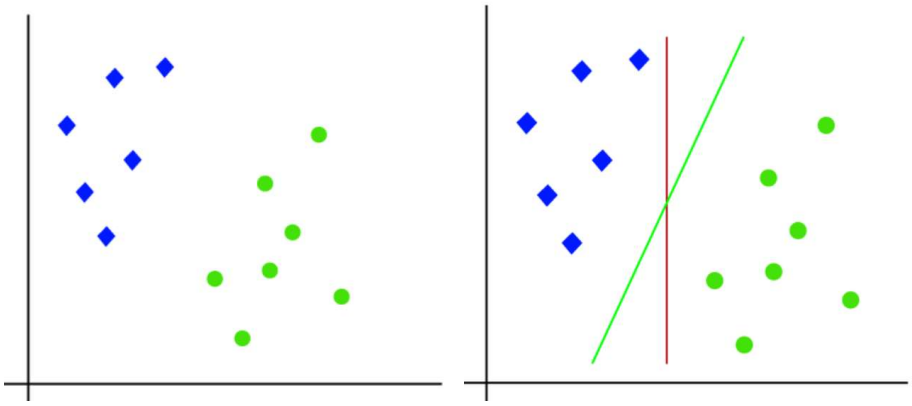
- The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector.

- Since these vectors support the hyperplane, hence called a Support vector.

- How does SVM works?

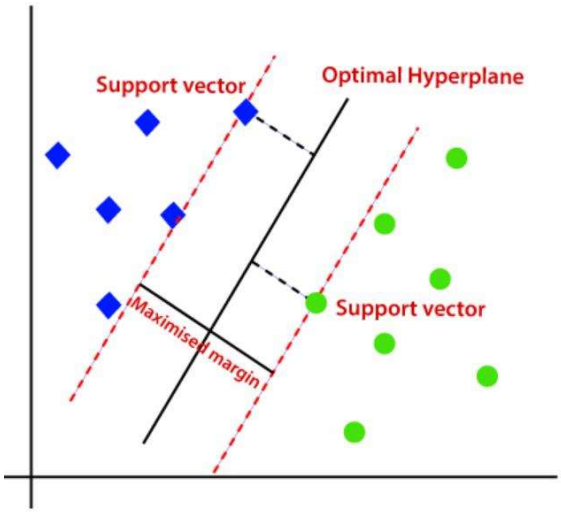
  Linear SVM:

- 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:

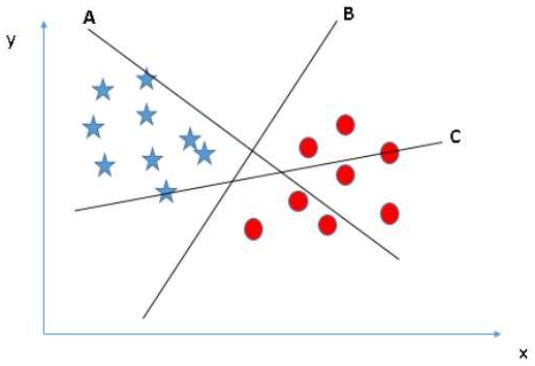


- So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes.

- 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**.

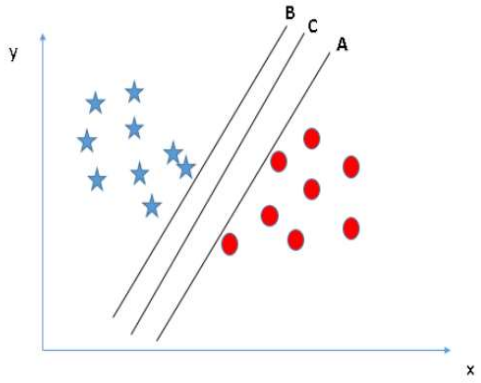


Identify the right hyper-plane (Scenario-1): Here, we have three hyper-planes (A, B, and C). Now, identify the right hyper-plane to classify stars and circles.

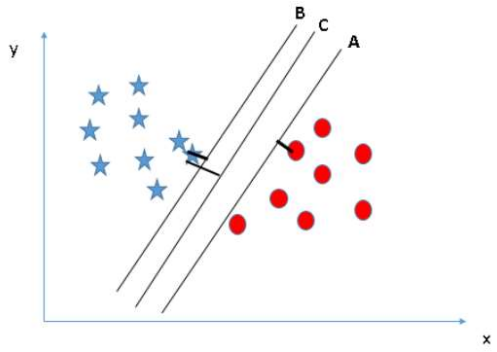


You need to remember a thumb rule to identify the right hyper-plane: "Select the hyper-plane which segregates the two classes better". In this scenario, hyper-plane "B" has excellently performed this job.

Identify the right hyper-plane (Scenario-2): Here, we have three hyper-planes (A, B, and C) and all are segregating the classes well. Now, How can we identify the right hyper-plane?

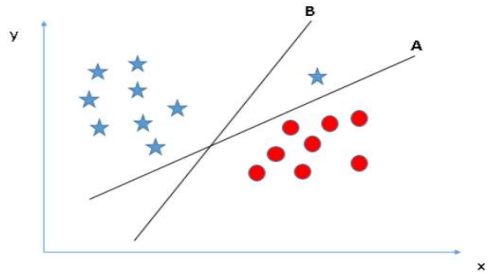


Here, maximizing the distances between nearest data point (either class) and hyper-plane will help us to decide the right hyper-plane. This distance is called as **Margin**. Let's look at the below snapshot:



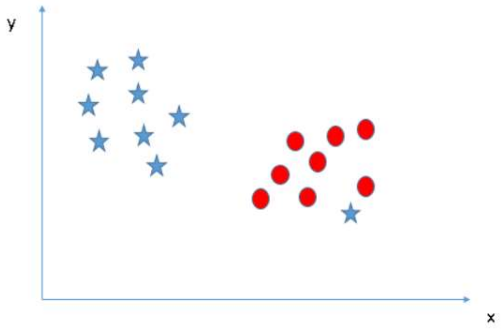
Above, you can see that the margin for hyper-plane C is high as compared to both A and B. Hence, we name the right hyper-plane as C. Another lightning reason for selecting the hyper-plane with higher margin is robustness. If we select a hyper-plane having low margin then there is high chance of miss-classification

Identify the right hyper-plane (Scenario-3): Hint: Use the rules as discussed in previous section to identify the right hyper-plane



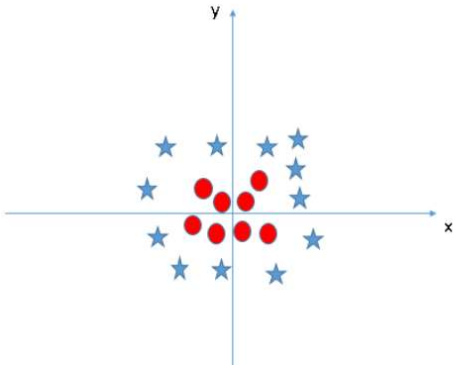
Some of you may have selected the hyper-plane **B** as it has higher margin compared to **A.** But, here is the catch, SVM selects the hyper-plane which classifies the classes accurately prior to maximizing margin. Here, hyper-plane B has a classification error and A has classified all correctly. Therefore, the right hyper-plane is **A.**

Can we classify two classes (Scenario-4)?: Below, I am unable to segregate the two classes using a straight line, as one of the stars lies in the territory of other(circle) class as an outlier.

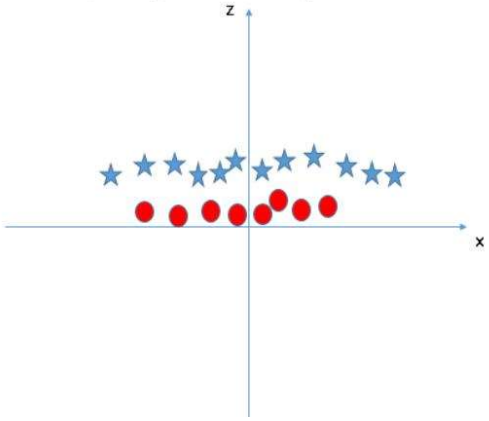


As I have already mentioned, one star at other end is like an outlier for star class. The SVM algorithm has a feature to ignore outliers and find the hyper-plane that has the maximum margin. Hence, we can say, SVM classification is robust to outliers.

Find the hyper-plane to segregate to classes (Scenario-5): In the scenario below, we can't have linear hyper-plane between the two classes, so how does SVM classify these two classes? Till now, we have only looked at the linear hyper-plane.



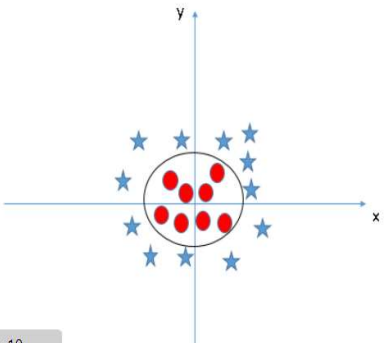
SVM can solve this problem. Easily! It solves this problem by introducing additional feature. Here, we will add a new feature z=x^2+y^2. Now, let's plot the data points on axis x and z:



- In above plot, points to consider are:
- All values for z would be positive always because z is the squared sum of both x and y

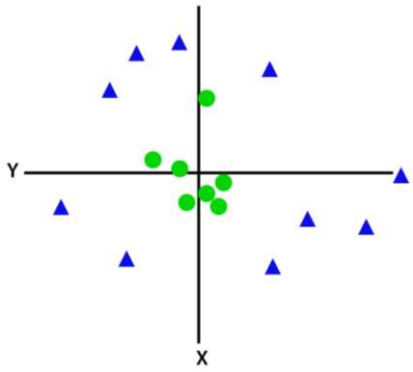
- In the original plot, red circles appear close to the origin of x and y axes, leading to lower value of z and star relatively away from the origin result to higher value of z.
- In the SVM classifier, it is easy to have a linear hyper-plane between these two classes.
- But, another burning question which arises is, should we need to add this feature manually to have a hyper-plane.
- No, the SVM algorithm has a technique called the **kernel trick**.
- The SVM kernel is a function that takes low dimensional input space and transforms it to a higher dimensional space i.e. it converts not separable problem to separable problem.
- It is mostly useful in non-linear separation problem. Simply put, it does some extremely complex data transformations, then finds out the process to separate the data based on the labels or outputs you've defined.

When we look at the hyper-plane in original input space it looks like a circle:



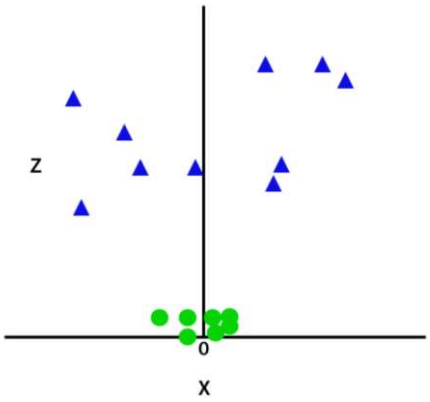
Non-Linear SVM:

- 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:

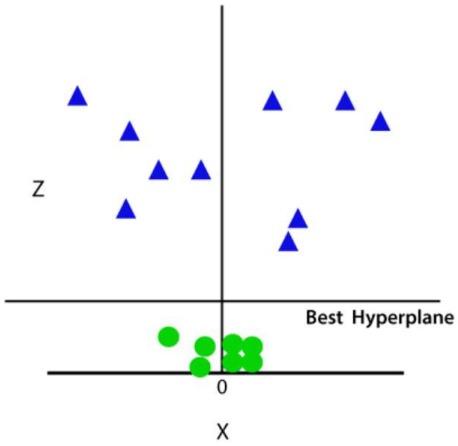


So to separate these data points, we need to add one more dimension.

- 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:
- 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:



7.Explain K-Nearest Neighbors and its working in detail

K-Nearest Neighbors (KNN) is a simple yet effective non-linear classification algorithm. It works by finding the k closest training examples to a new observation and assigning the most common class label among these neighbors.

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

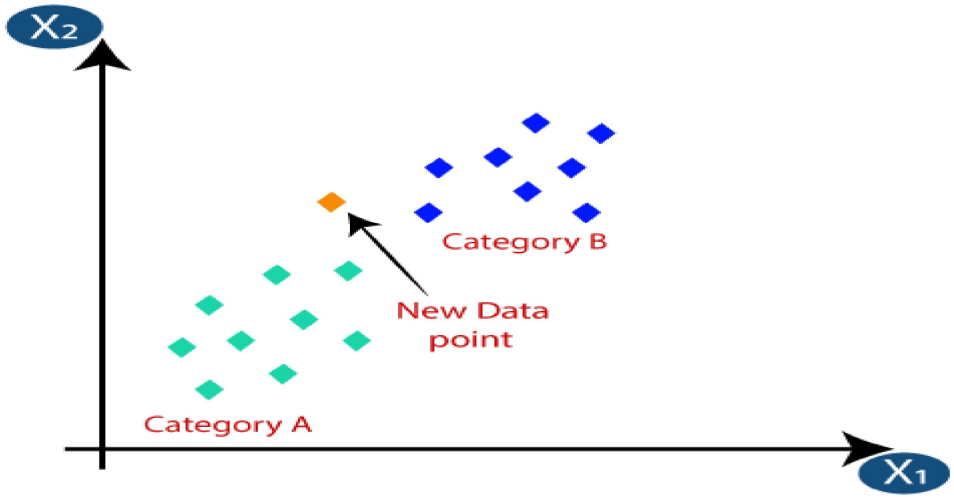
Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:

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

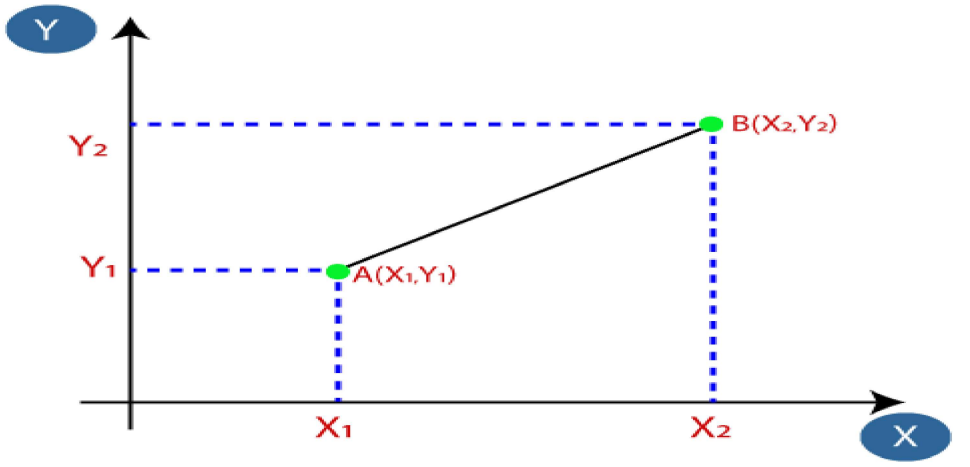
- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.

- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

Suppose we have a new data point and we need to put it in the required category. Consider the below image:

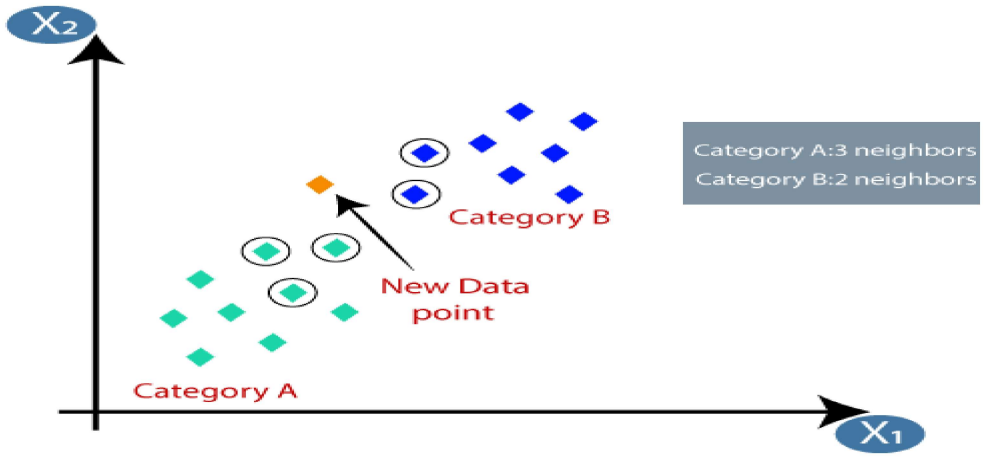


- Firstly, we will choose the number of neighbors, so we will choose the k=5.
- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



$$\text{Euclidean Distance between } A_1 \text{ and } B_2 = \sqrt{(X_2-X_1)^2+(Y_2-Y_1)^2}$$

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

- 

As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

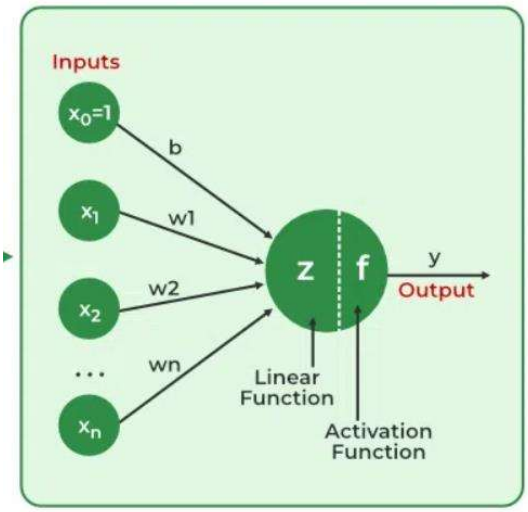
8.Explain working of Neural Network.

Neural Networks extract identifying features from data, lacking pre-programmed understanding. Network components include neurons, connections, weights, biases, propagation functions, and a learning rule. Neurons receive inputs, governed by thresholds and activation functions. Connections involve weights and biases regulating information transfer. Learning, adjusting weights and biases, occurs in three stages: input computation, output generation, and iterative refinement enhancing the network's proficiency in diverse tasks.

An *artificial neural network* (ANN) is a machine learning model inspired by the structure and function of the human brain's interconnected network of neurons. It consists of interconnected nodes called artificial neurons, organized into layers. Information flows through the network, with each neuron processing input signals and producing an output signal that influences other neurons in the network.

These include:

1. The neural network is simulated by a new environment.
2. Then the free parameters of the neural network are changed as a result of this simulation.
3. The neural network then responds in a new way to the environment because of the changes in its free parameters.



Importance of Neural Networks

The ability of neural networks to identify patterns, solve intricate puzzles, and adjust to changing surroundings is essential. Their capacity to learn from data has far-reaching effects, ranging from revolutionizing technology like natural language processing and self-driving automobiles to automating decision-making processes and increasing efficiency in numerous industries. The development of artificial intelligence is largely dependent on neural networks, which also drive innovation and influence the direction of technology.

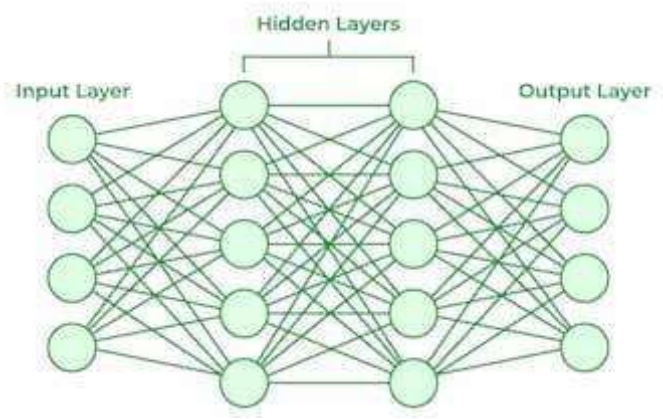
How does Neural Networks work?

Let's understand with an example of how a neural network works:

Consider a neural network for email classification. The input layer takes features like email content, sender information, and subject. These inputs, multiplied by adjusted weights, pass through hidden layers. The network, through training, learns to recognize patterns indicating whether an email is spam or not. The output layer, with a binary activation function, predicts whether the email is spam (1) or not (0). As the network iteratively refines its weights through backpropagation, it becomes adept at distinguishing between spam and legitimate emails, showcasing the practicality of neural networks in real-world applications like email filtering.

Working of a Neural Network

Neural networks are complex systems that mimic some features of the functioning of the human brain. It is composed of an input layer, one or more hidden layers, and an output layer made up of layers of artificial neurons that are coupled. The two stages of the basic process are called backpropagation and forward propagation.



Forward Propagation

- **Input Layer:** Each feature in the input layer is represented by a node on the network, which receives input data.
- **Weights and Connections:** The weight of each neuronal connection indicates how strong the connection is. Throughout training, these weights are changed.
- **Hidden Layers:** Each hidden layer neuron processes inputs by multiplying them by weights, adding them up, and then passing them through an activation function. By doing this, non-linearity is introduced, enabling the network to recognize intricate patterns.
- **Output:** The final result is produced by repeating the process until the output layer is reached.

Backpropagation

- **Loss Calculation:** The network's output is evaluated against the real goal values, and a loss function is used to compute the difference. For a regression problem, the Mean Squared Error (MSE) is commonly used as the cost function.

  Loss Function:

- **Gradient Descent:** Gradient descent is then used by the network to reduce the loss. To lower the inaccuracy, weights are changed based on the derivative of the loss with respect to each weight.
- **Adjusting weights:** The weights are adjusted at each connection by applying this iterative process, or backpropagation, backward across the network.
- **Training:** During training with different data samples, the entire process of forward propagation, loss calculation, and backpropagation is done iteratively, enabling the network to adapt and learn patterns from the data.
- **Actvation Functions:** Model non-linearity is introduced by activation functions like the rectified linear unit (ReLU) or sigmoid. Their decision on whether to "fire" a neuron is based on the whole weighted input.

9.Discuss briefly about CART in decision trees.

- **CART (Classification and Regression Tree)** is a variation of the decision tree algorithm. It can handle both classification and regression tasks.

- Scikit-Learn use the Classification and Regression Tree (CART) algorithm to train Decision Trees (also called “growing” trees).

CART Algorithm

- CART is a predictive algorithm used in Machine learning and it explains how the target variable’s values can be predicted based on other matters.

- It is a decision tree where each fork is split into a predictor variable and each node has a prediction for the target variable at the end.

- In the decision tree, nodes are split into sub-nodes on the basis of a threshold value of an attribute.

- The root node is taken as the training set and is split into two by considering the best attribute and threshold value. Further, the subsets are also split using the same logic.

- This continues till the last pure sub-set is found in the tree or the maximum number of leaves possible in that growing tree.

The CART algorithm works via the following process:

- The best split point of each input is obtained.

- Based on the best split points of each input in Step 1, the new “best” split point is identified.

- Split the chosen input according to the “best” split point.

- Continue splitting until a stopping rule is satisfied or no further desirable splitting is available.

Gini index/Gini impurity

- The Gini index is a metric for the classification tasks in CART. It stores the sum of squared probabilities of each class.

- It computes the degree of probability of a specific variable that is wrongly being classified when chosen randomly and a variation of the Gini coefficient.

- It works on categorical variables, provides outcomes either “successful” or “failure” and hence conducts binary splitting only.

- The degree of the Gini index varies from 0 to 1,

- Where 0 depicts that all the elements are allied to a certain class or only one class exists there.
- The Gini index of value 1 signifies that all the elements are randomly distributed across various classes, and
- A value of 0.5 denotes the elements are uniformly distributed into some classes.

Mathematically, we can write Gini Impurity as follows:

$$Gini = 1 - \sum_{i=1}^{n} (pi)^2$$

where pi is the probability of an object being classified to a particular class.

Classification tree

• A classification tree is an algorithm where the target variable is categorical. The algorithm is then used to identify the "Class" within which the target variable is most likely to fall. Classification trees are used when the dataset needs to be split into classes that belong to the response variable(like yes or no)

Regression tree

• A Regression tree is an algorithm where the target variable is continuous and the tree is used to predict its value. Regression trees are used when the response variable is continuous. For example, if the response variable is the temperature of the day.

CART model representation

- CART models are formed by picking input variables and evaluating split points on those variables until an appropriate tree is produced.

Steps to create a Decision Tree using the CART algorithm:

- ***Greedy algorithm***: In this the input space is divided using the Greedy method which is known as a recursive binary spitting.
- This is a numerical method within which all of the values are aligned and several other split points are tried and assessed using a cost function.
- ***Stopping Criterion:*** As it works its way down the tree with the training data, the recursive binary splitting method described above must know when to stop splitting.
- The most frequent halting method is to utilize a minimum amount of training data allocated to every leaf node. If the count is smaller than the specified threshold, the split is rejected and also the node is considered the last leaf node.

- ***Tree pruning:*** Decision tree's complexity is defined as the number of splits in the tree. Trees with fewer branches are recommended as they are simple to grasp and less prone to cluster the data.

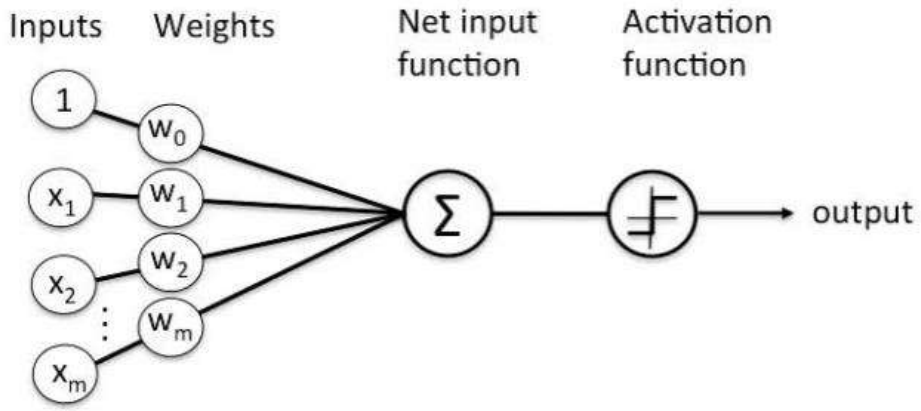
- ***Data preparation for the CART*:** No special data preparation is required for the CART algorithm.

10.Explain multi-layer perceptron architecture.

A ***multi-layer perceptron (MLP)*** is a type of artificial neural network consisting of multiple layers of neurons. The neurons in the MLP typically use nonlinear activation functions, allowing the network to learn complex patterns in data. MLPs are significant in machine learning because they can learn nonlinear relationships in data, making them powerful models for tasks such as classification, regression, and pattern recognition.

Perceptron:

Perceptron was introduced by *Frank Rosenblatt* in 1957. A Perceptron is an algorithm for supervised learning of binary classifiers. This algorithm enables neurons to learn and processes elements in the training set one at a time.



A Perceptron

Multilayer Perceptron:

A multilayer perceptron is a type of feedforward neural network consisting of fully connected neurons with a nonlinear kind of activation function. It is widely used to distinguish data that is not linearly separable.

MLPs have been widely used in various fields, including image recognition, natural language processing, and speech recognition, among others. Their flexibility in architecture and ability to approximate any function under certain conditions make them a fundamental building block in deep learning and neural network research. Some of its key concepts are as follows:

Input layer:

The input layer consists of nodes or neurons that receive the initial input data. Each neuron represents a feature or dimension of the input data. The number of neurons in the input layer is determined by the dimensionality of the input data.

Hidden layer:

Between the input and output layers, there can be one or more layers of neurons. Each neuron in a hidden layer receives inputs from all neurons in the previous layer (either the input layer or another hidden layer) and produces an output that is passed to the next layer. The number of hidden layers

and the number of neurons in each hidden layer are hyperparameters that need to be determined during the model design phase.

Output layer:

This layer consists of neurons that produce the final output of the network. The number of neurons in the output layer depends on the nature of the task. In binary classification, there may be either one or two neurons depending on the activation function and representing the probability of belonging to one class; while in multi-class classification tasks, there can be multiple neurons in the output layer.

Weights:

Neurons in adjacent layers are fully connected to each other. Each connection has an associated weight, which determines the strength of the connection. These weights are learned during the training process.

Bias Neurons:

In addition to the input and hidden neurons, each layer (except the input layer) usually includes a bias neuron that provides a constant input to the neurons in the next layer. The bias neuron has its own weight associated with each connection, which is also learned during training.

The bias neuron effectively shifts the activation function of the neurons in the subsequent layer, allowing the network to learn an offset or bias in the decision boundary. By adjusting the weights connected to the bias neuron, the MLP can learn to control the threshold for activation and better fit the training data.

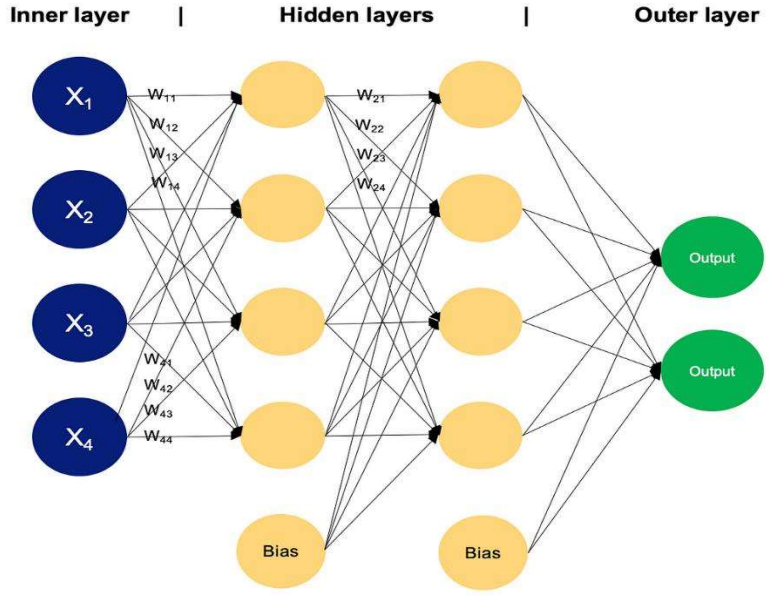
Activation Function:

Typically, each neuron in the hidden layers and the output layer applies an activation function to its weighted sum of inputs. Common activation functions include sigmoid, tanh, ReLU (Rectified Linear Unit), and softmax. These functions introduce nonlinearity into the network, allowing it to learn complex patterns in the data.

Training with Backpropagation:

MLPs are trained using the backpropagation algorithm, which computes gradients of a loss function with respect to the model's parameters and updates the parameters iteratively to minimize the loss.

Layer by Layer Working of a Multilayer Perceptron:



Example of a MLP having two hidden layers

In a multilayer perceptron, neurons process information in a step-by-step manner, performing computations that involve weighted sums and nonlinear transformations. Let's walk layer by layer to see the magic that goes within.

Input layer:

- The input layer of an MLP receives input data, which could be features extracted from the input samples in a dataset. Each neuron in the input layer represents one feature.
- Neurons in the input layer do not perform any computations; they simply pass the input values to the neurons in the first hidden layer.

Hidden layers:

- The hidden layers of an MLP consist of interconnected neurons that perform computations on the input data.
- Each neuron in a hidden layer receives input from all neurons in the previous layer. The inputs are multiplied by corresponding weights, denoted as w. The weights determine how much influence the input from one neuron has on the output of another.
- In addition to weights, each neuron in the hidden layer has an associated bias, denoted as b. The bias provides an additional input to the neuron, allowing it to adjust its output threshold. Like weights, biases are learned during training.
- For each neuron in a hidden layer or the output layer, the weighted sum of its inputs is computed. This involves multiplying each input by its corresponding weight, summing up these products, and adding the bias:

$$\text{Weighted Sum} = \sum_{i=1}^{n} (w_i * x_i) + b$$

Where n is the total number of input connections, wi is the weight for the i-th input, and xi is the i-th input value.

- The weighted sum is then passed through an activation function, denoted as f. The activation function introduces nonlinearity into the network, allowing it to learn and represent complex relationships in the data. The activation function determines the output range of the neuron and its behavior in response to different input values. The choice of activation function depends on the nature of the task and the desired properties of the network.

Output layer:

- The output layer of an MLP produces the final predictions or outputs of the network. The number of neurons in the output layer depends on the task being performed (e.g., binary classification, multi-class classification, regression).
- Each neuron in the output layer receives input from the neurons in the last hidden layer and applies an activation function. This activation function is usually different from those used in the hidden layers and produces the final output value or prediction.

During the training process, the network learns to adjust the weights associated with each neuron's inputs to minimize the discrepancy between the predicted outputs and the true target values in the training data. By adjusting the weights and learning the appropriate activation functions, the network learns to approximate complex patterns and relationships in the data, enabling it to make accurate predictions on new, unseen samples.

This adjustment is guided by an optimization algorithm, such as stochastic gradient descent (SGD), which computes the gradients of a loss function with respect to the weights and updates the weights iteratively.