

## Supervised Learning

Training a model-Linear Regression, Multiple Linear regression, Improving accuracy of Linear Regression Model, Polynomial Regression Model Classification-Introduction, Decision Tree, Random Forest Model, Support Vector Machines, Boosting

### Regression Analysis

- Regression analysis is a statistical method to model the relationship between a dependent (target) and independent (predictor) variables with one or more independent variables.
- Regression analysis helps us to understand how the value of the dependent variable is changing corresponding to an independent variable when other independent variables are held fixed. It predicts continuous/real values such as **temperature, age, salary, price**, etc.
- Regression is a supervised learning technique which helps in finding the correlation between variables and enables us to predict the continuous output variable based on the one or more predictor variables. It is mainly used for **prediction, forecasting, time series modeling, and determining the causal-effect relationship between variables**.
- In Regression, we plot a graph between the variables which best fits the given datapoints, using this plot, the machine learning model can make predictions about the data.
- ***"Regression shows a line or curve that passes through all the datapoints on target-predictor graph in such a way that the vertical distance between the datapoints and the regression line is minimum."***
- The distance between datapoints and line tells whether a model has captured a strong relationship or not.

We can understand the concept of regression analysis using the below example:

**Example:** Suppose there is a marketing company A, who does various advertisement every year and get sales on that. The below list shows the advertisement made by the company in the last 5 years and the corresponding sales.

Now, the company wants to do the advertisement of \$200 in the year 2019 **and wants to know the prediction about the sales for this year**. So to solve such type of prediction problems in machine learning, we need regression analysis.

Advertisement	Sales
\$90	\$1000
\$120	\$1300
\$150	\$1800
\$100	\$1200
\$130	\$1380
\$200	??

**Some examples of regression can be as:**

- Prediction of rain using temperature and other factors
- Determining Market trends
- Prediction of road accidents due to rash driving.

**Terminologies Related to the Regression Analysis:**

- **Dependent Variable:** The main factor in Regression analysis which we want to predict or understand is called the dependent variable. It is also called **target variable**.

- **Independent Variable:** The factors which affect the dependent variables or which are used to predict the values of the dependent variables are called independent variable, also called as a **predictor**.
- **Outliers:** Outlier is an observation which contains either very low value or very high value in comparison to other observed values. An outlier may hamper the result, so it should be avoided.
- **Multicollinearity:** If the independent variables are highly correlated with each other than other variables, then such condition is called Multicollinearity. It should not be present in the dataset, because it creates problem while ranking the most affecting variable.
- **Underfitting and Overfitting:** If our algorithm works well with the training dataset but not well with test dataset, then such problem is called **Overfitting**. And if our algorithm does not perform well even with training dataset, then such problem is called **underfitting**.

### Why do we use Regression Analysis?

Regression analysis helps in the prediction of a continuous variable. There are various scenarios in the real world where we need some future predictions such as weather condition, sales prediction, marketing trends, etc., for such case we need some technology which can make predictions more accurately. So for such case we need Regression analysis which is a statistical method and used in machine learning and data science. Below are some other reasons for using Regression analysis:

- Regression estimates the relationship between the target and the independent variable.
- It is used to find the trends in data.
- It helps to predict real/continuous values.
- By performing the regression, we can confidently determine the **most important factor, the least important factor, and how each factor is affecting the other factors**.

### Types of Regression

There are various types of regressions which are used in data science and machine learning. Each type has its own importance on different scenarios, but at the core, all the regression methods analyze the effect of the independent variable on dependent variables. Here we are discussing some important types of regression which are given below:

1. Linear Regression
2. Logistic Regression
3. Polynomial Regression
4. Support Vector Regression
5. Decision Tree Regression
6. Random Forest Regression
7. Ridge Regression
8. Lasso Regression

#### 1. Linear Regression:

- Linear regression is a statistical regression method which is used for predictive analysis.
- It is one of the very simple and easy algorithms which works on regression and shows the relationship between the continuous variables.
- It is used for solving the regression problem in machine learning.

- Linear regression shows the linear relationship between the independent variable (X-axis) and the dependent variable (Y-axis), hence called linear regression.
- If there is only one input variable (x), then such linear regression is called **simple linear regression**. And if there is more than one input variable, then such linear regression is called **multiple linear regression**.
- The relationship between variables in the linear regression model can be explained using the below image. Here we are predicting the salary of an employee on the basis of **the year of experience**.

Below is the mathematical equation for Linear regression:

$$Y = aX + b$$

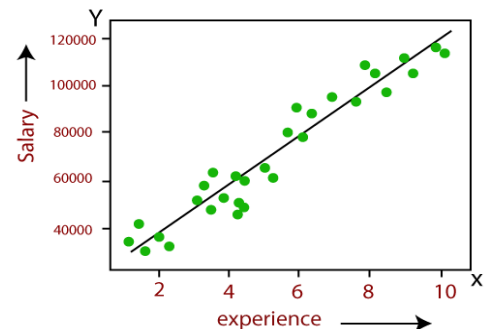
Here, Y = dependent variables (target variables),

X = Independent variables (predictor variables),

a and b are the linear coefficients

**Some popular applications of linear regression are:**

- Analyzing trends and sales estimates
- Salary forecasting
- Real estate prediction
- Arriving at ETAs in traffic.



## 2. Logistic Regression:

- Logistic regression is another supervised learning algorithm which is used to solve the classification problems. In **classification problems**, we have dependent variables in a binary or discrete format such as 0 or 1.
- Logistic regression algorithm works with the categorical variable such as 0 or 1, Yes or No, True or False, Spam or not spam, etc.
- It is a predictive analysis algorithm which works on the concept of probability.
- Logistic regression is a type of regression, but it is different from the linear regression algorithm in the term how they are used.
- Logistic regression uses **sigmoid function** or logistic function which is a complex cost function. This sigmoid function is used to model the data in logistic regression. The function can be represented as:

$$f(x) = \frac{1}{1 + e^{-x}}$$

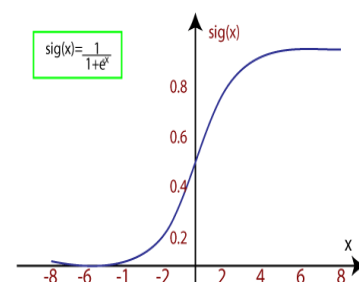
f(x) = Output between the 0 and 1 value.

x = input to the function

e = base of natural logarithm.

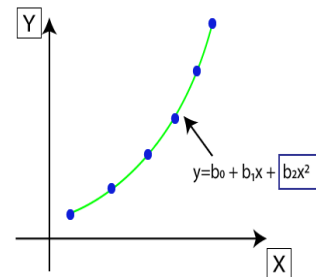
When we provide the input values (data) to the function, it gives the S-curve as follows:

- It uses the concept of threshold levels, values above the threshold level are rounded up to 1, and values below the threshold level are rounded up to 0.
- There are three types of logistic regression:
  - ✓ **Binary(0/1, pass/fail)**
  - ✓ **Multi(cats, dogs, lions)**
  - ✓ **Ordinal(low, medium, high)**



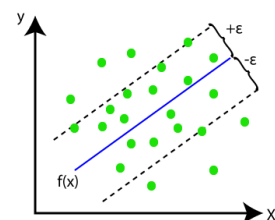
## 3. Polynomial Regression:

- Polynomial Regression is a type of regression which models the **non-linear dataset** using a linear model.
- It is similar to multiple linear regression, but it fits a non-linear curve between the value of  $x$  and corresponding conditional values of  $y$ .
- Suppose there is a dataset which consists of datapoints which are present in a non-linear fashion, so for such case, linear regression will not best fit to those datapoints. To cover such datapoints, we need Polynomial regression.
- In **Polynomial regression, the original features are transformed into polynomial features of given degree and then modeled using a linear model**. Which means the datapoints are best fitted using a polynomial line.
- The equation for polynomial regression also derived from linear regression equation that means Linear regression equation  $Y = b_0 + b_1x$ , is transformed into Polynomial regression equation  $Y = b_0 + b_1x + b_2x^2 + b_3x^3 + \dots + b_nx^n$ .
- Here  $Y$  is the **predicted/target output**,  $b_0, b_1, \dots, b_n$  are the **regression coefficients**.  $x$  is our **independent/input variable**.
- The model is still linear as the coefficients are still linear with quadratic



#### 4. Support Vector Regression:

- Support Vector Machine is a supervised learning algorithm which can be used for regression as well as classification problems. So if we use it for regression problems, then it is termed as Support Vector Regression.
- Support Vector Regression is a regression algorithm which works for continuous variables. Below are some keywords which are used in **Support Vector Regression**:
  - ✓ **Kernel**: It is a function used to map a lower-dimensional data into higher dimensional data.
  - ✓ **Hyperplane**: In general SVM, it is a separation line between two classes, but in SVR, it is a line which helps to predict the continuous variables and cover most of the datapoints.
  - ✓ **Boundary line**: Boundary lines are the two lines apart from hyperplane, which creates a margin for datapoints.
  - ✓ **Support vectors**: Support vectors are the datapoints which are nearest to the hyperplane and opposite class.

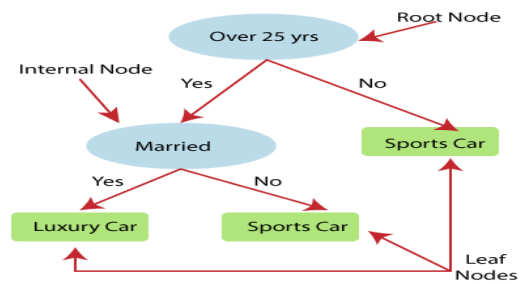


- In SVR, we always try to determine a hyperplane with a maximum margin, so that maximum number of datapoints are covered in that margin.
- **The main goal of SVR is to consider the maximum datapoints within the boundary lines and the hyperplane (best-fit line) must contain a maximum number of datapoints.**
- Consider the image, the blue line is called hyperplane, and the other two lines are known as boundary lines.

#### 5. Decision Tree Regression:

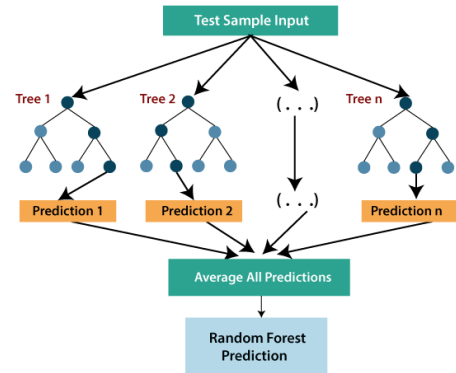
- Decision Tree is a supervised learning algorithm which can be used for solving both classification and regression problems.
- It can solve problems for both categorical and numerical data
- Decision Tree regression builds a tree-like structure in which each internal node represents the "test" for an attribute, each branch represent the result of the test, and each leaf node represents the final decision or result.

- A decision tree is constructed starting from the root node/parent node (dataset), which splits into left and right child nodes (subsets of dataset).
- These child nodes are further divided into their children node, and themselves become the parent node of those nodes.
- The image showing the example of Decision Tree regression, here, the model is trying to predict the choice of a person between Sports cars or Luxury car.



## 6. Random forest

- Random forest is one of the most powerful supervised learning algorithms which is capable of performing regression as well as classification tasks.
- The Random Forest regression is an ensemble learning method which combines multiple decision trees and predicts the final output based on the average of each tree output. The combined decision trees are called as base models, and it can be represented more formally as:



- $g(x) = f_0(x) + f_1(x) + f_2(x) + \dots$
- Random forest uses **Bagging or Bootstrap Aggregation** technique of ensemble learning in which aggregated decision tree runs in parallel and do not interact with each other.
- With the help of Random Forest regression, we can prevent Overfitting in the model by creating random subsets of the dataset.

## 9. Ridge Regression:

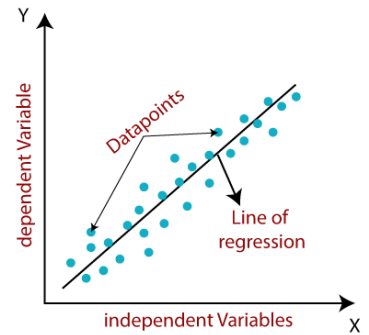
- Ridge regression is one of the most robust versions of linear regression in which a small amount of bias is introduced so that we can get better long term predictions.
- The amount of bias added to the model is known as **Ridge Regression penalty**. We can compute this penalty term by multiplying with the lambda to the squared weight of each individual features.
- The equation for ridge regression will be:

$$L(x, y) = \text{Min} \left( \sum_{i=1}^n (y_i - w_i x_i)^2 + \lambda \sum_{i=1}^n (w_i)^2 \right)$$

- A general linear or polynomial regression will fail if there is high collinearity between the independent variables, so to solve such problems, Ridge regression can be used.
- Ridge regression is a regularization technique, which is used to reduce the complexity of the model. It is also called as **L2 regularization**.
- It helps to solve the problems if we have more parameters than samples.

### Lasso Regression:

- Lasso regression is another regularization technique to reduce the complexity of the model.
- It is similar to the Ridge Regression except that penalty term contains only the absolute weights instead of a square of weights.
- Since it takes absolute values, hence, it can shrink the slope to 0, whereas Ridge Regression can only shrink it near to 0.
- It is also called as **L1 regularization**. The equation for Lasso regression will be:



$$L(x, y) = \text{Min}(\sum_{i=1}^n (y_i - w_i x_i)^2 + \lambda \sum_{i=1}^n |w_i|)$$

## Representation of Linear Regression

Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (x) 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

$$y = a_0 + a_1x + \epsilon$$

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).		
$\epsilon$	= random error			

The values for x and y variables are training datasets for Linear Regression model representation.

## Types of Linear Regression

Linear regression can be further divided into two types of the algorithm:

### 1. SimpleLinearRegression:

If a single independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Simple Linear Regression.

### 2. MultipleLinearRegression:

If more than one independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Multiple Linear Regression.

## 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.

**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.

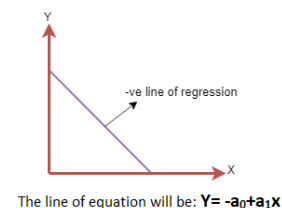
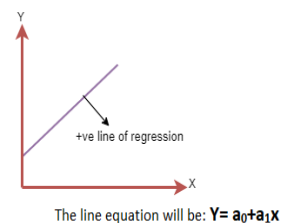
### Finding the best fit line:

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**.
- For Linear Regression, we use the **Mean Squared Error (MSE)** cost function, which is the average of squared error occurred between the predicted values and actual values. It can be written as:
- For the above linear equation, MSE can be calculated as:



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

Where,

$N$  = Total number of observation value  
 $y_i$  = Actual value  
 $(a_1x_i + a_0)$  = Predicted value.

**Residuals:** The distance between the actual value and predicted values is called residual. If the observed points are far from the regression line, then the residual will be high, and so

cost function will high. If the scatter points are close to the regression line, then the residual will be small and hence the cost function.

### Training a model-Simple Linear Regression

Simple Linear Regression is a type of Regression algorithms that models the relationship between a dependent variable and a single independent variable. The relationship shown by a Simple Linear Regression model is linear or a sloped straight line, hence it is called Simple Linear Regression.

The key point in Simple Linear Regression is that the ***dependent variable must be a continuous/real value***. However, the independent variable can be measured on continuous or categorical values.

Simple Linear regression algorithm has mainly two objectives:

- **Model the relationship between the two variables.** Such as the relationship between Income and expenditure, experience and Salary, etc.
- **Forecasting new observations.** Such as Weather forecasting according to temperature, Revenue of a company according to the investments in a year, etc.

#### ❖ Implementation of Simple Linear Regression Algorithm using Python

- create a Simple Linear Regression model to find out the best fitting line for representing the relationship between these two variables.
- Taking a dataset that has two variables: salary (dependent variable) and experience (Independent variable).

	YearsExperience,Salary
1	1.1,39343.00
2	1.3,46205.00
3	1.5,37731.00
4	2.0,43525.00
5	2.2,39891.00
6	2.9,56642.00
7	3.0,60150.00
8	3.2,54445.00
9	3.2,64445.00
10	3.7,57189.00
11	3.9,63218.00
12	4.0,55794.00
13	4.0,56957.00
14	4.1,57081.00
15	4.5,61111.00
16	4.9,67938.00
17	5.1,66029.00
18	5.3,83088.00
19	5.9,81363.00
20	6.0,93940.00
21	6.8,91738.00
22	7.1,98273.00
23	7.9,101302.00
24	8.2,113812.00
25	8.7,109431.00
26	9.0,105582.00
27	9.5,116969.00
28	9.6,112635.00
29	10.3,122391.00
30	10.5,121872.00
31	

Steps for Implementation:

#### Step 1.Data Pre-processing:

- Import the three important libraries, which will help us for loading the dataset, plotting the graphs, and creating the Simple Linear Regression model.
- Next,load the dataset into our code
- Extract the dependent and independent variables from the given dataset. The independent variable is years of experience, and the dependent variable is salary.
- 
- Next,split both variables into the test set and training set. We have 30 observations, so we will take 20 observations for the training set and 10 observations for the test set.

```
import numpy as nm
```

```
import matplotlib.pyplot as mtp
```

```
import pandas as pd
```

```
data_set= pd.read_csv('data.csv')
```

```
data_set
```



```
x= data_set.iloc[:, :-1].values
y= data_set.iloc[:, 1].values
# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 1/3, random_state=0)
```

## Step 2: Fitting the Simple Linear Regression to the Training Set:

To fit our model to the training dataset. To do so, we will import the **LinearRegression** class of the **linear\_model** library from the **scikit learn**. After importing the class, we are going to create an object of the class named as a **regressor**.

```
from sklearn.linear_model import LinearRegression

regressor= LinearRegression()

regressor.fit(x_train, y_train)
```

a **fit()** method to fit our Simple Linear Regression object to the training set. In the **fit()** function, we have passed the **x\_train** and **y\_train**, which is our training dataset for the dependent and an independent variable. We have fitted our regressor object to the training set so that the model can easily learn the correlations between the predictor and target variables. After executing the above lines of code, we will get the below output.

---

```
Out[37]: LinearRegression()
```

---

## Step: 3. Prediction of test set result:

our model is ready to predict the output for the new observations. In this step, we will provide the test dataset (new observations) to the model to check whether it can predict the correct output or not.

We will create a prediction vector **y\_pred**, and **x\_pred**, which will contain predictions of test dataset, and prediction of training set respectively.

## #Prediction of Test and Training set result

```
y_pred= regressor.predict(x_test)
x_pred= regressor.predict(x_train)
```

## Step: 4. visualizing the Training set results:

- In this step, visualize the training set result. use the **scatter()** function of the **pyplot** library, which we have already imported in the pre-processing step. The **scatter () function** will create a scatter plot of observations.
- In the x-axis-plot the Years of Experience of employees.
- The y-axis- plot the salary of employees.
- In the **scatter () function**, we will pass the real values of training set, which means a year of experience **x\_train**, training set of Salaries **y\_train**, and color of the observations. Here we are taking a green color for the observation, but it can be any color as per the choice.
- plot the regression line, use **plot() function** of the **pyplot** library. In this function, we will pass the years of experience for training set, predicted salary for training set **x\_pred**, and color of the line.

- title for the plot. use the **title()** function of the **pyplot** library and pass the name ("Salary vs Experience (Training Dataset)").
- After that, Assign labels for x-axis and y-axis using **xlabel()** and **ylabel()** function.

```
mtp.scatter(x_train, y_train, color="green")
mtp.plot(x_train, x_pred, color="red")
```



```
mtp.title("Salary vs Experience (Training Dataset)")
mtp.xlabel("Years of Experience")
mtp.ylabel("Salary(In Rupees)")
mtp.show()
```

By executing the above lines of code, we will get the below graph plot as an output.

In the above plot, we can see the real values observations in green dots and predicted values are covered by the red regression line. The regression line shows a correlation between the dependent and independent variable.

The good fit of the line can be observed by calculating the difference between actual values and predicted values. But as we can see in the above plot, most of the observations are close to the regression line, hence our model is good for the training set.

#### Step: 5. visualizing the Test set results:

In the previous step, we have visualized the performance of our model on the training set. Now, we will do the same for the Test set. The complete code will remain the same as the above code, except in this, we will use **x\_test**, and **y\_test** instead of **x\_train** and **y\_train**.

Here we are also changing the color of observations and regression line to differentiate between the two plots, but it is optional.

#### #visualizing the Test set results

```
mtp.scatter(x_test, y_test, color="blue")
mtp.plot(x_train, x_pred, color="red")
mtp.title("Salary vs Experience (Test Dataset)")
mtp.xlabel("Years of Experience")
mtp.ylabel("Salary(In Rupees)")
mtp.show()
```



By executing the above line of code, we will get the output as:

In the plot, there are observations given by the blue color, and prediction is given by the red regression line. As we can see, most of the observations are close to the regression line, hence we can say our Simple Linear Regression is a good model and able to make good predictions.

## Training a model-Multiple Linear Regression

Multiple Linear Regression is an extension of Simple Linear regression as it takes more than one predictor variable to predict the response variable. We can define it as:

*Multiple Linear Regression is one of the important regression algorithms which models the linear relationship between a single dependent continuous variable and more than one independent variable.*

### Example:

Prediction of CO<sub>2</sub> emission based on engine size and number of cylinders in a car.

### Some key points about MLR:

For MLR, the dependent or target variable(Y) must be the continuous/real, but the predictor or independent variable may be of continuous or categorical form.

Each feature variable must model the linear relationship with the dependent variable.

MLR tries to fit a regression line through a multidimensional space of data-points.

MLR equation:

In Multiple Linear Regression, the target variable(Y) is a linear combination of multiple predictor variables  $x_1, x_2, x_3, \dots, x_n$ . Since it is an enhancement of Simple Linear Regression, so the same is applied for the multiple linear regression equation, the equation becomes:

$$Y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \dots + b_nx_n \quad \dots\dots\dots (a)$$

Where,

**Y= Output/Response variable**

**$b_0, b_1, b_2, b_3, b_n, \dots$  = Coefficients of the model.**

**$x_1, x_2, x_3, x_4, \dots$  = Various Independent/feature variable**

Assumptions for Multiple Linear Regression:

A **linear relationship** should exist between the Target and predictor variables.

The regression residuals must be **normally distributed**.

MLR assumes little or **no multicollinearity** (correlation between the independent variable) in data.

Implementation of Multiple Linear Regression model using Python:

To implement MLR using Python, we have below problem:

### Problem Description:

We have a dataset of **50 start-up companies**. This dataset contains five main information: **R&D Spend, Administration Spend, Marketing Spend, State, and Profit for a financial year**. Our goal is to create a model that can easily determine which company has a maximum profit, and which is the most affecting factor for the profit of a company.

Since we need to find the Profit, so it is the dependent variable, and the other four variables are independent variables. Below are the main steps of deploying the MLR model:

### Data Pre-processing Steps

#### Fitting the MLR model to the training set

#### Predicting the result of the test set

#### Step-1: Data Pre-processing Step:

The very first step is data pre-processing, which we have already discussed in this tutorial. This process contains the below steps:

**Importing libraries:** Firstly we will import the library which will help in building the model. Below is the code for it:

```
# importing libraries
```

```
import numpy as nm
```

```
import matplotlib.pyplot as mtp
```

```
import pandas as pd
```

**Importing dataset:** Now we will import the dataset(50\_ComplList), which contains all the variables. Below is the code for it:

```
#importing datasets
```

```
data_set= pd.read_csv('50_ComplList.csv')
```

**Output:** We will get the dataset as:

data\_set - DataFrame

Index	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349	136898	471784	New York	192262
1	162598	151378	443899	California	191792
2	153442	101146	407935	Florida	191050
3	144372	118672	383200	New York	182902
4	142107	91391.8	366168	Florida	166188
5	131877	99814.7	362861	New York	156991
6	134615	147199	127717	California	156123
7	130298	145530	323877	Florida	155753
8	120543	148719	311613	New York	152212
9	123335	108679	304982	California	149760
10	101913	110594	229161	Florida	146122
11	100672	91790.6	249745	California	144259
12	93863.8	127320	249839	Florida	141586
13	91992.4	135495	252665	California	134307

Format    Resize    ☒ Background color    ☒ Column min/max    Save and Close    Close

In above output, we can clearly see that there are five variables, in which four variables are continuous and one is categorical variable.

**Extracting dependent and independent Variables:**

```
#Extracting Independent and dependent Variable
```

```
x= data_set.iloc[:, :-1].values
```

```
y= data_set.iloc[:, 4].values
```

**Output:**

**Out[5]:**

```
array([[165349.2, 136897.8, 471784.1, 'New York'],  
       [162597.7, 151377.59, 443898.53, 'California'],  
       [153441.51, 101145.55, 407934.54, 'Florida'],  
       [144372.41, 118671.85, 383199.62, 'New York'],  
       [142107.34, 91391.77, 366168.42, 'Florida'],  
       [131876.9, 99814.71, 362861.36, 'New York'],  
       [134615.46, 147198.87, 127716.82, 'California'],  
       [130298.13, 145530.06, 323876.68, 'Florida'],  
       [120542.52, 148718.95, 311613.29, 'New York'],  
       [123334.88, 108679.17, 304981.62, 'California'],  
       [101913.08, 110594.11, 229160.95, 'Florida'],  
       [100671.96, 91790.61, 249744.55, 'California'],  
       [93863.75, 127320.38, 249839.44, 'Florida'],  
       [91992.39, 135495.07, 252664.93, 'California'],  
       [119943.24, 156547.42, 256512.92, 'Florida'],  
       [114523.61, 122616.84, 261776.23, 'New York'],  
       [78013.11, 121597.55, 264346.06, 'California'],  
       [94657.16, 145077.58, 282574.31, 'New York'],  
       [91749.16, 114175.79, 294919.57, 'Florida'],  
       [86419.7, 153514.11, 0.0, 'New York'],  
       [76253.86, 113867.3, 298664.47, 'California'],  
       [78389.47, 153773.43, 299737.29, 'New York'],  
       [73994.56, 122782.75, 303319.26, 'Florida'],  
       [67532.53, 105751.03, 304768.73, 'Florida'],  
       [77044.01, 99281.34, 140574.81, 'New York'],  
       [64664.71, 139553.16, 137962.62, 'California'],  
       [75328.87, 144135.98, 134050.07, 'Florida'],  
       [72107.6, 127864.55, 353183.81, 'New York'],  
       [66051.52, 182645.56, 118148.2, 'Florida'],
```

```
[65605.48, 153032.06, 107138.38, 'New York'],
[61994.48, 115641.28, 91131.24, 'Florida'],
[61136.38, 152701.92, 88218.23, 'New York'],
[63408.86, 129219.61, 46085.25, 'California'],
[55493.95, 103057.49, 214634.81, 'Florida'],
[46426.07, 157693.92, 210797.67, 'California'],
[46014.02, 85047.44, 205517.64, 'New York'],
[28663.76, 127056.21, 201126.82, 'Florida'],
[44069.95, 51283.14, 197029.42, 'California'],
[20229.59, 65947.93, 185265.1, 'New York'],
[38558.51, 82982.09, 174999.3, 'California'],
[28754.33, 118546.05, 172795.67, 'California'],
[27892.92, 84710.77, 164470.71, 'Florida'],
[23640.93, 96189.63, 148001.11, 'California'],
[15505.73, 127382.3, 35534.17, 'New York'],
[22177.74, 154806.14, 28334.72, 'California'],
[1000.23, 124153.04, 1903.93, 'New York'],
[1315.46, 115816.21, 297114.46, 'Florida'],
[0.0, 135426.92, 0.0, 'California'],
[542.05, 51743.15, 0.0, 'New York'],
[0.0, 116983.8, 45173.06, 'California']], dtype=object)
```

As we can see in the above output, the last column contains categorical variables which are not suitable to apply directly for fitting the model. So we need to encode this variable.

### Encoding Dummy Variables:

As we have one categorical variable (State), which cannot be directly applied to the model, so we will encode it. To encode the categorical variable into numbers, we will use the **LabelEncoder** class. But it is not sufficient because it still has some relational order, which may create a wrong model. So in order to remove this problem, we will use **OneHotEncoder**, which will create the dummy variables. Below is code for it:

```
#Categorical data

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

labelencoder_x= LabelEncoder()

x[:, 3]= labelencoder_x.fit_transform(x[:,3])

onehotencoder= OneHotEncoder(categorical_features= [3])

x= onehotencoder.fit_transform(x).toarray()
```

Here we are only encoding one independent variable, which is state as other variables are continuous.

### Output:

x - NumPy array

	0	1	2	3	4	5
0	0	0	1	165349	136898	471784
1	1	0	0	162598	151378	443899
2	0	1	0	153442	101146	407935
3	0	0	1	144372	118672	383200
4	0	1	0	142107	91391.8	366168
5	0	0	1	131877	99814.7	362861
6	1	0	0	134615	147199	127717
7	0	1	0	130298	145530	323877
8	0	0	1	120543	148719	311613
9	1	0	0	123335	108679	304982
10	0	1	0	101913	110594	229161
11	1	0	0	100672	91790.6	249745
12	0	1	0	93863.8	127320	249839
13	1	0	0	91992.4	135495	252665

Format    Resize    ☒ Background color

As we can see in the above output, the state column has been converted into dummy variables (0 and 1). **Here each dummy variable column is corresponding to the one State.** We can check by comparing it with the original dataset. The first column corresponds to the **California State**, the second column corresponds to the **Florida State**, and the third column corresponds to the **New York State**.

Note: We should not use all the dummy variables at the same time, so it must be 1 less than the total number of dummy variables, else it will create a dummy variable trap.

Now, we are writing a single line of code just to avoid the dummy variable trap:

#avoiding the dummy variable trap:

```
x = x[:, 1:]
```

If we do not remove the first dummy variable, then it may introduce multicollinearity in the model.

x - NumPy array

	0	1	2	3	4
0	0	1	165349	136898	471784
1	0	0	162598	151378	443899
2	1	0	153442	101146	407935
3	0	1	144372	118672	383200
4	1	0	142107	91391.8	366168
5	0	1	131877	99814.7	362861
6	0	0	134615	147199	127717
7	1	0	130298	145530	323877
8	0	1	120543	148719	311613
9	0	0	123335	108679	304982
10	1	0	101913	110594	229161
11	0	0	100672	91790.6	249745
12	1	0	93863.8	127320	249839

Format    Resize    ☒ Background color

Save and Close    Close

As we can see in the above output image, the first column has been removed.

Now we will split the dataset into training and test set. The code for this is given below:

# Splitting the dataset into training and test set.

```
from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.2, random_state=0)
```

The above code will split our dataset into a training set and test set.

**Output:** The above code will split the dataset into training set and test set. You can check the output by clicking on the variable explorer option given in Spyder IDE. The test set and training set will look like the below image:

**Test set:**



y\_test - NumPy array

	0
0	103282
1	144259
2	146122
3	77798.8
4	191050
5	105008
6	81229.1
7	97483.6
8	110352
9	166188

Format
Resize
☒ Background color

x\_test - NumPy array

	0	1	2	3	4
0	1	0	66051.5	182646	118148
1	0	0	100672	91790.6	249745
2	1	0	101913	110594	229161
3	1	0	27892.9	84710.8	164471
4	1	0	153442	101146	407935
5	0	1	72107.6	127865	353184
6	0	1	20229.6	65947.9	185265
7	0	1	61136.4	152702	88218.2
8	1	0	73994.6	122783	303319
9	1	0	142107	91391.8	366168

Format
Resize
☒ Background color

Save and Close
Close

## Training set:

x\_train - NumPy array

	0	1	2	3	4
0	1	0	55493.9	103057	214635
1	0	1	46014	85047.4	205518
2	1	0	75328.9	144136	134050
3	0	0	46426.1	157694	210798
4	1	0	91749.2	114176	294920
5	1	0	130298	145530	323877
6	1	0	119943	156547	256513
7	0	1	1000.23	124153	1903.93
8	0	1	542.05	51743.2	0
9	0	1	65605.5	153032	107138
10	0	1	114524	122617	261776
11	1	0	61994.5	115641	91131.2

Format
Resize
☒ Background color

Save and Close
Close

y\_train - NumPy array

	0
0	96778.9
1	96479.5
2	105734
3	96712.8
4	124267
5	155753
6	132603
7	64926.1
8	35673.4
9	101005
10	129917
11	99937.6

Format
Resize
☒ Background color

Save and Close
Close

Note: In MLR, we will not do feature scaling as it is taken care by the library, so we don't need to do it manually.

Step: 2- Fitting our MLR model to the Training set:

Now, we have well prepared our dataset in order to provide training, which means we will fit our regression model to the training set. It will be similar to as we did in Simple Linear Regression model. The code for this will be:

#Fitting the MLR model to the training set:

```
from sklearn.linear_model import LinearRegression
```

```
regressor= LinearRegression()  
regressor.fit(x_train, y_train)
```

### Output:

Out[9]: LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=None, normalize=False)

Now, we have successfully trained our model using the training dataset. In the next step, we will test the performance of the model using the test dataset.

### Step: 3- Prediction of Test set results:

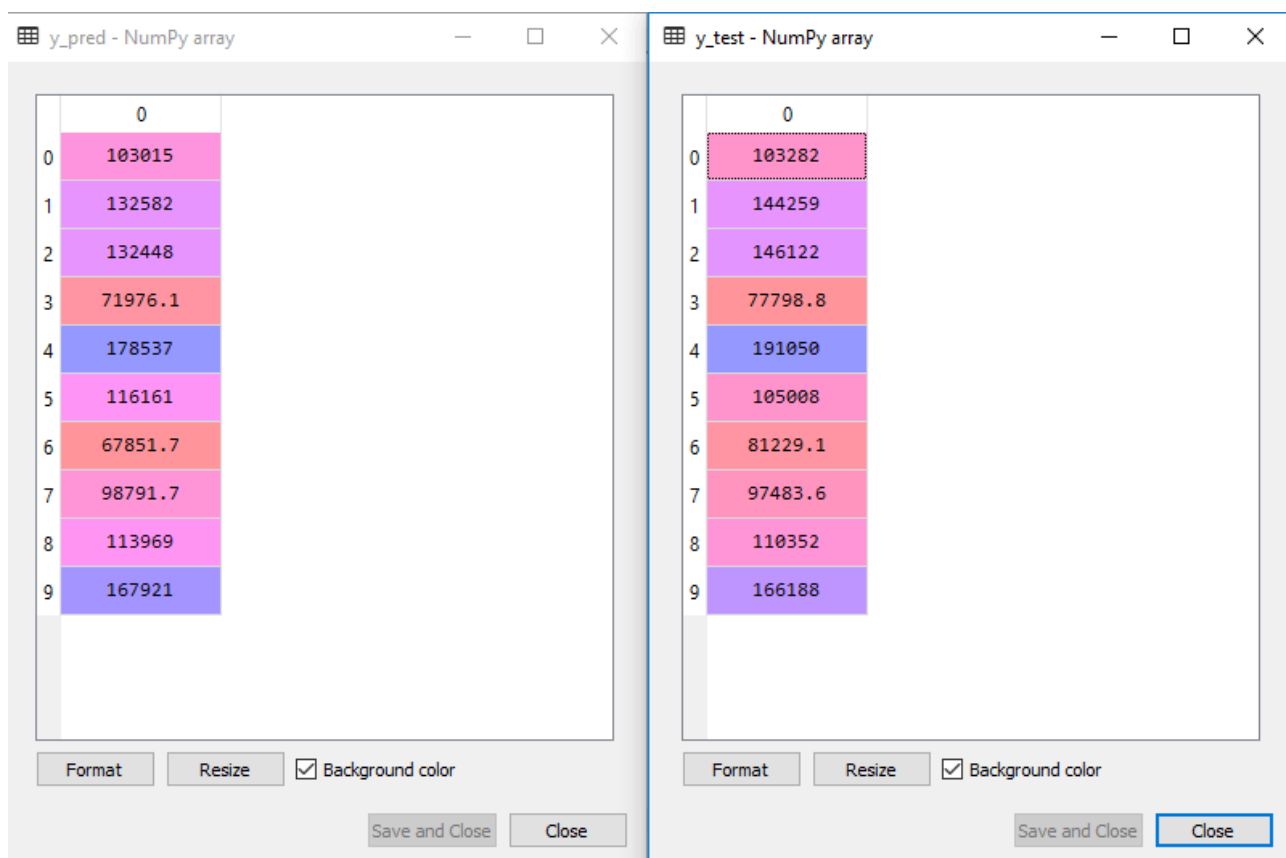
The last step for our model is checking the performance of the model. We will do it by predicting the test set result. For prediction, we will create a **y\_pred** vector. Below is the code for it:

```
#Predicting the Test set result;
```

```
y_pred= regressor.predict(x_test)
```

By executing the above lines of code, a new vector will be generated under the variable explorer option. We can test our model by comparing the predicted values and test set values.

### Output:



In the above output, we have predicted result set and test set. We can check model performance by comparing these two value index by index. For example, the first index has a predicted value of **103015\$** profit and test/real value of **103282\$** profit. The difference is only of **267\$**, which is a good prediction, so, finally, our model is completed here.

We can also check the score for training dataset and test dataset. Below is the code for it:

```
print('Train Score: ', regressor.score(x_train, y_train))
```

```
print('Test Score: ', regressor.score(x_test, y_test))
```

**Output:** The score is:

Train Score: 0.9501847627493607

Test Score: 0.9347068473282446

**The above score tells that our model is 95% accurate with the training dataset and 93% accurate with the test dataset.**

#### **Applications of Multiple Linear Regression:**

- Effectiveness of Independent variable on prediction:
- Predicting the impact of changes

### **Improving accuracy of Linear Regression Model**

There are several ways to improve the accuracy of a linear regression model:

**Feature selection:** Choosing the right set of features (independent variables) is important for a good model. Unnecessary or irrelevant features can introduce noise and reduce the accuracy of the model. Feature selection techniques such as forward selection, backward elimination, and stepwise regression can be used to identify the most relevant features.

**Regularization:** Regularization techniques such as L1 (Lasso) and L2 (Ridge) regularization can be used to prevent overfitting and improve the accuracy of the model. Regularization adds a penalty term to the cost function that encourages the model to have smaller weights and reduces the complexity of the model.

**Data cleaning:** Cleaning and preprocessing the data can also improve the accuracy of the model. This involves removing outliers, handling missing values, and normalizing or standardizing the data.

**Cross-validation:** Cross-validation techniques such as k-fold cross-validation can be used to evaluate the performance of the model on different subsets of the data. This can help to identify and prevent overfitting.

**Ensemble methods:** Ensemble methods such as bagging and boosting can be used to improve the accuracy of the model. Bagging involves training multiple models on different subsets of the data and combining their predictions, while boosting involves iteratively training models that focus on the misclassified data points.

**Non-linear transformations:** In some cases, transforming the data using non-linear functions can improve the accuracy of the model. For example, transforming the independent variables using a logarithmic or exponential function can help to capture non-linear relationships between the variables.

Overall, improving the accuracy of a linear regression model involves a combination of feature selection, regularization, data cleaning, cross-validation, ensemble methods, and non-linear transformations. It is important to experiment with different techniques and evaluate the performance of the model on different subsets of the data.

#### **Polynomial Regression:**

- Polynomial Regression is a regression algorithm that models the relationship between a dependent(y) and independent variable(x) as nth degree polynomial. The Polynomial Regression equation is given below:

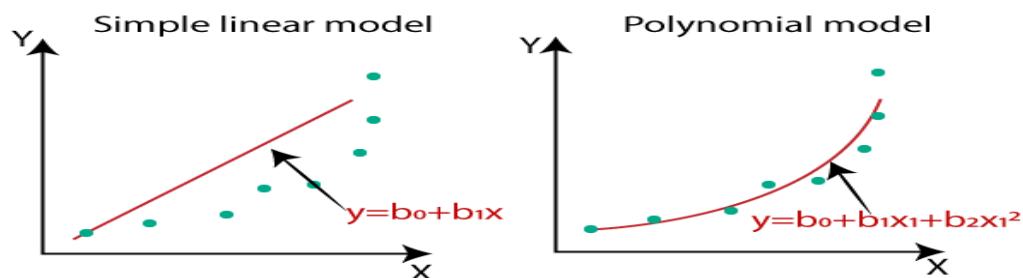
$$y = b_0 + b_1x_1 + b_2x_1^2 + b_3x_1^3 + \dots + b_nx_1^n$$

- It is also called the special case of Multiple Linear Regression in ML. Because we add some polynomial terms to the Multiple Linear regression equation to convert it into Polynomial Regression.

- It is a linear model with some modification in order to increase the accuracy.
- The dataset used in Polynomial regression for training is of non-linear nature.
- It makes use of a linear regression model to fit the complicated and non-linear functions and datasets.
- **Hence, "In Polynomial regression, the original features are converted into Polynomial features of required degree (2,3,..,n) and then modeled using a linear model."**

#### Need for Polynomial Regression:

- If we apply a linear model on a **linear dataset**, then it provides us a good result as we have seen in Simple Linear Regression, but if we apply the same model without any modification on a **non-linear dataset**, then it will produce a drastic output. Due to which loss function will increase, the error rate will be high, and accuracy will be decreased.
- So for such cases, **where data points are arranged in a non-linear fashion, we need the Polynomial Regression model**. We can understand it in a better way using the below comparison diagram of the linear dataset and non-linear dataset.



- In the above image, we have taken a dataset which is arranged non-linearly. So if we try to cover it with a linear model, then we can clearly see that it hardly covers any data point. On the other hand, a curve is suitable to cover most of the data points, which is of the Polynomial model.
- Hence, *if the datasets are arranged in a non-linear fashion, then we should use the Polynomial Regression model instead of Simple Linear Regression.*

#### Equation of the Polynomial Regression Model:

**Simple Linear Regression equation:**  $y = b_0 + b_1x$  .....(a)

**Multiple Linear Regression equation:**  $y = b_0 + b_1x + b_2x_2 + b_3x_3 + \dots + b_nx_n$  .....(b)

**Polynomial Regression equation:**  $y = b_0 + b_1x + b_2x^2 + b_3x^3 + \dots + b_nx^n$  .....(c)

all three equations are Polynomial equations but differ by the degree of variables. The Simple and Multiple Linear equations are also Polynomial equations with a single degree, and the Polynomial regression equation is Linear equation with the nth degree. So if we add a degree to our linear equations, then it will be converted into Polynomial Linear equations.

# CLASSIFICATION IN MACHINE LEARNING:

The Classification algorithm is a Supervised Learning technique that is used to identify the category of new observations on the basis of training data. In Classification, a program learns from the given dataset or observations and then classifies new observation into a number of classes or groups. Such as, **Yes or No, 0 or 1, Spam or Not Spam, cat or dog**, etc. Classes can be called as targets/labels or categories.

Unlike regression, the output variable of Classification is a category, not a value, such as "Green or Blue", "fruit or animal", etc. Since the Classification algorithm is a Supervised learning technique, hence it takes labeled input data, which means it contains input with the corresponding output.

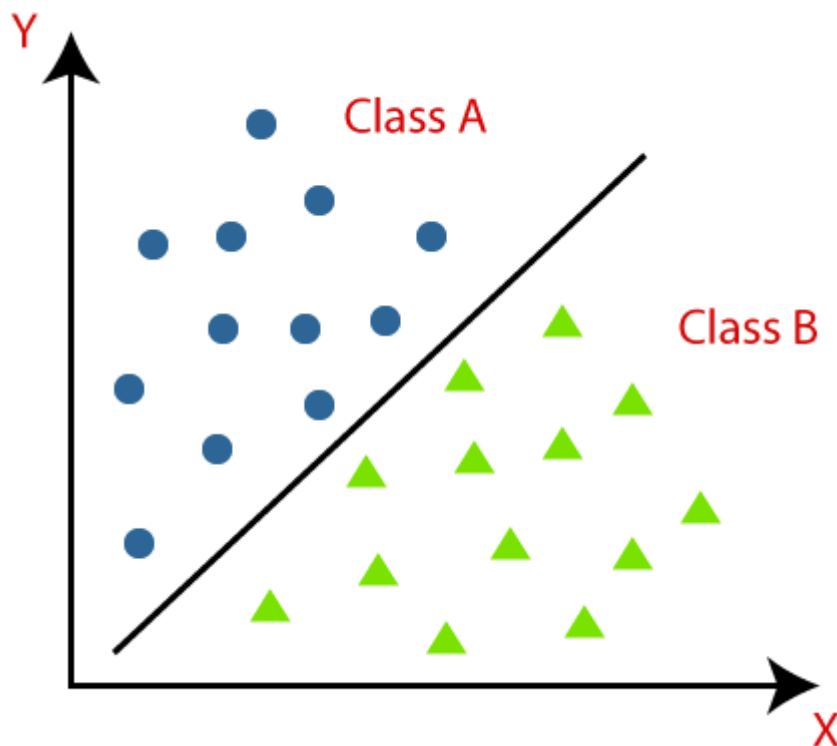
In classification algorithm, a discrete output function( $y$ ) is mapped to input variable( $x$ ).

1.  $y=f(x)$ , where  $y$  = categorical output

The best example of an ML classification algorithm is **Email Spam Detector**.

The main goal of the Classification algorithm is to identify the category of a given dataset, and these algorithms are mainly used to predict the output for the categorical data.

Classification algorithms can be better understood using the below diagram. In the below diagram, there are two classes, class A and Class B. These classes have features that are similar to each other and dissimilar to other classes.



The algorithm which implements the classification on a dataset is known as a classifier. There are two types of Classifications:

- **Binary Classifier:** If the classification problem has only two possible outcomes, then it is called as Binary Classifier.  
**Examples:** YES or NO, MALE or FEMALE, SPAM or NOT SPAM, CAT or DOG, etc.
- **Multi-class Classifier:** If a classification problem has more than two outcomes, then called as Multi-class Classifier  
**Example:** Classifications of types of crops, Classification of types of music.

## Types of ML Classification Algorithms:

Classification Algorithms can be further divided into the Mainly two category:

- **Linear Models** ○ Logistic Regression ○ Support Vector Machines
- **Non-linear Models** ○ K-Nearest Neighbours ○ Kernel SVM ○ Naïve Bayes ○ Decision Tree Classification ○ Random Forest Classification

## Use cases of Classification Algorithms

Classification algorithms can be used in different places. Below are some popular use cases of Classification Algorithms:

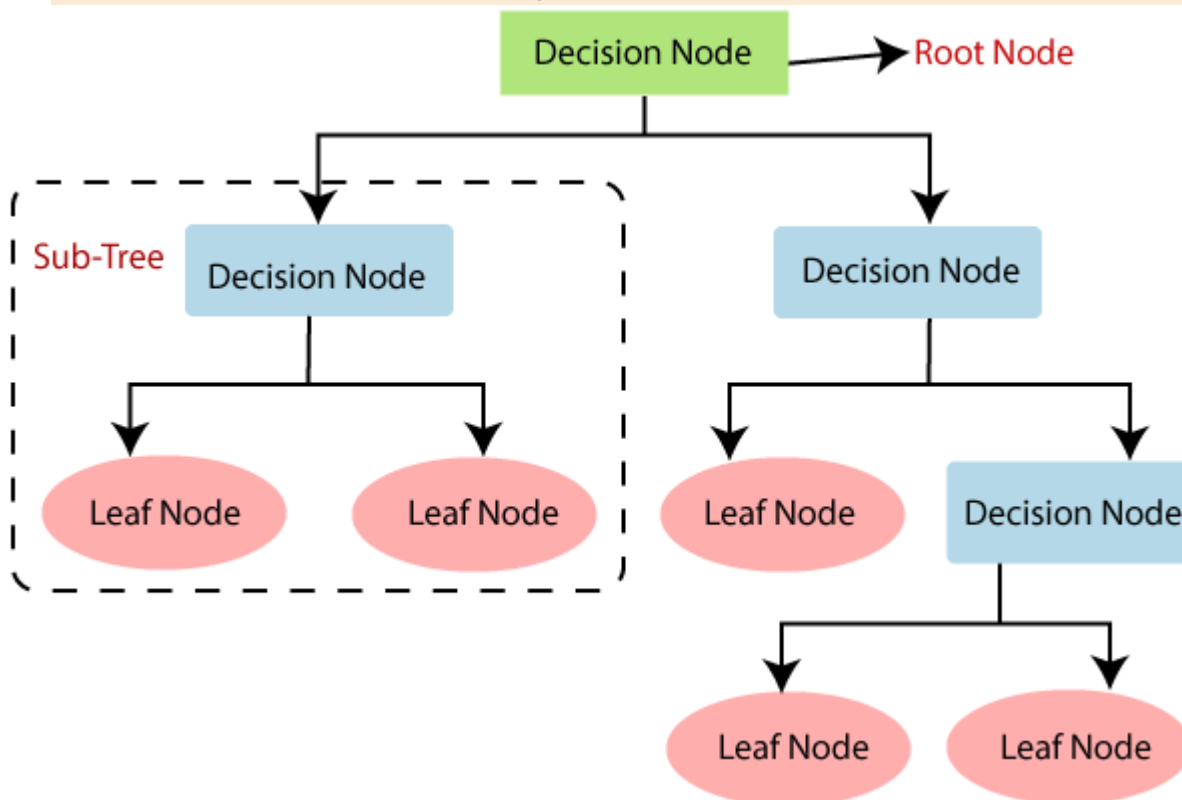
- Email Spam Detection
- Speech Recognition ○ Identifications of Cancer tumor cells.
- Drugs Classification ○ Biometric Identification, etc.

## Decision Tree Classification Algorithm

- Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree structured classifier, where **internal nodes represent the features of a dataset**, **branches represent the decision rules** and **each leaf node represents the outcome**.

- In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node**. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
- The decisions or the test are performed on the basis of features of the given dataset. ○ ***It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.***
- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
- In order to build a tree, we use the **CART algorithm**, which stands for **Classification and Regression Tree algorithm**.
- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.
- Below diagram explains the general structure of a decision tree:

***Note: A decision tree can contain categorical data (YES/NO) as well as numeric data.***



## Why use Decision Trees?

There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model. Below are the two reasons for using the Decision tree:

- Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
- The logic behind the decision tree can be easily understood because it shows a tree-like structure.

# Decision Tree Terminologies

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

## How does the Decision Tree algorithm Work?

In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

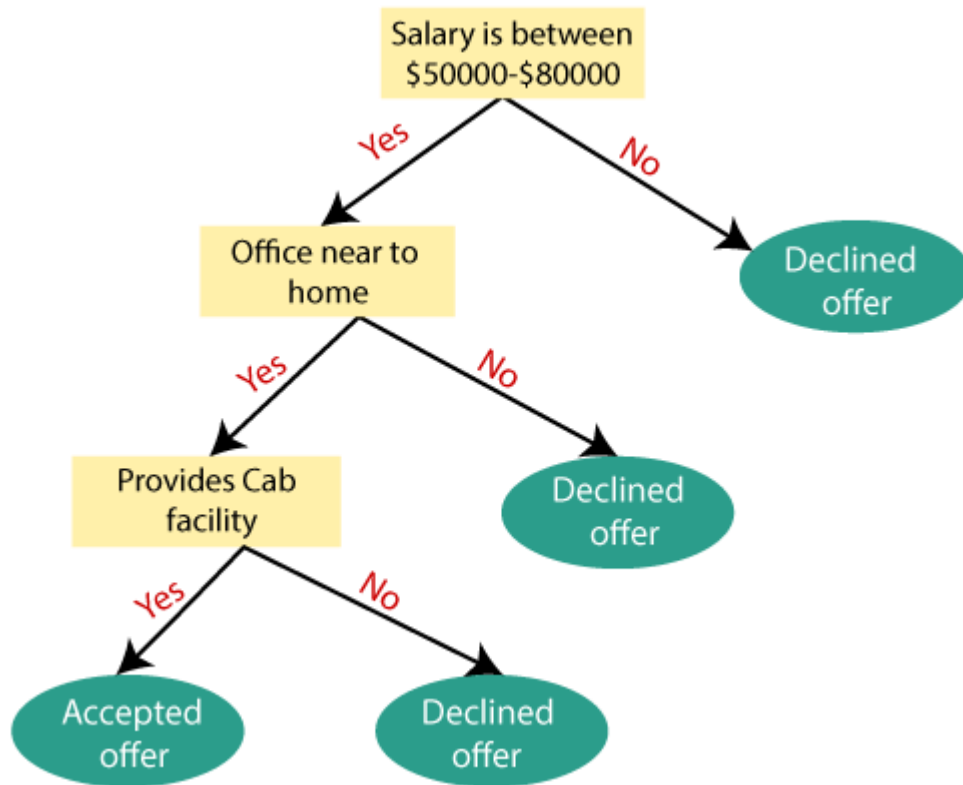
For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm.

- **Step-1:** Begin the tree with the root node, says S, which contains the complete dataset.
- **Step-2:** Find the best attribute in the dataset using **Attribute Selection Measure (ASM)**.
- **Step-3:** Divide the S into subsets that contains possible values for the best attributes.
- **Step-4:** Generate the decision tree node, which contains the best attribute.
- **Step-5:** Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

**Example:** Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not. So, to solve this problem, the decision tree starts with the root node (Salary attribute by

ASM). The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels. The next decision node further gets split into one decision node (Cab facility) and one leaf node. Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer). Consider the below diagram:





## Attribute Selection Measures

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

- **Information Gain**
- **Gini Index**

### 1. Information Gain:

- Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.
- It calculates how much information a feature provides us about a class.
- According to the value of information gain, we split the node and build the decision tree. ○ A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:

$$1. \text{ Information Gain} = \text{Entropy}(S) - [(\text{Weighted Avg}) * \text{Entropy}(\text{each feature})]$$

**Entropy:** Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

$$\text{Entropy}(s) = -P(\text{yes}) \log_2 P(\text{yes}) - P(\text{no}) \log_2 P(\text{no})$$

Where,

○ **S= Total number**

**of samples** ○

**P(yes)= probability**

**of yes** ○ **P(no)=**

**probability of no**

**2. Gini Index:** ○ Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.

○ An attribute with the low Gini index should be preferred as compared to the high Gini

index. ○ It only creates binary splits, and the CART algorithm uses the Gini index to

create binary splits. ○ Gini index can be calculated using the below formula:

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

## Pruning: Getting an Optimal Decision tree

*Pruning is a process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree.*

A too-large tree increases the risk of overfitting, and a small tree may not capture all the important features of the dataset. Therefore, a technique that decreases the size of the learning tree without reducing accuracy is known as Pruning. There are mainly two types of tree **pruning** technology used:

○ **Cost Complexity**

**Pruning** ○

**Reduced Error**

**Pruning.**

## Advantages of the Decision Tree

○ It is simple to understand as it follows the same process which a human follow while making any decision in real-life.

○ It can be very useful for solving decision-related problems. ○ It helps to think about all the possible outcomes for a problem. ○ There is less requirement of data cleaning compared to other algorithms.

# Disadvantages of the Decision Tree

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

## Python Implementation of Decision Tree

Now we will implement the Decision tree using Python. For this, we will use the dataset "user\_data.csv," which we have used in previous classification models. By using the same dataset, we can compare the Decision tree classifier with other classification models such as [KNN](#) [SVM](#), [LogisticRegression](#), etc.

Steps will also remain the same, which are given below:

- **Data Pre-processing step** ○ **Fitting a Decision-Tree algorithm to the Training set** ○ **Predicting the test result** ○ **Test accuracy of the result(Creation of Confusion matrix)** ○ **Visualizing the test set result.**

### 1. Data Pre-Processing Step:

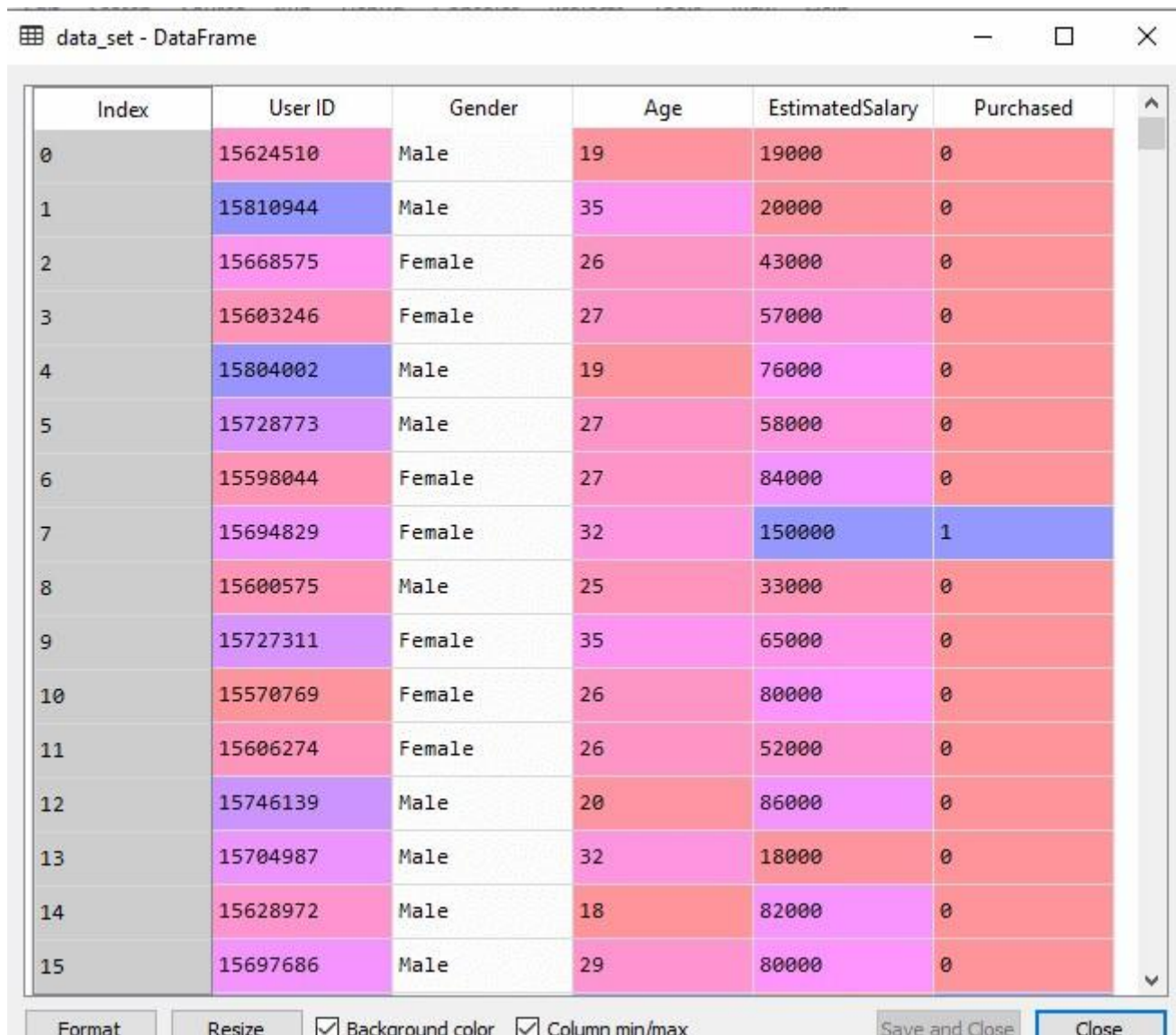
Below is the code for the pre-processing step:

1. # importing libraries
2. **import** numpy as nm
3. **import** matplotlib.pyplot as mtp
4. **import** pandas as pd
- 5.
6. #importing datasets
7. data\_set= pd.read\_csv('user\_data.csv')
- 8.
9. #Extracting Independent and dependent Variable
10. x= data\_set.iloc[:, [2,3]].values
11. y= data\_set.iloc[:, 4].values
- 12.
13. # Splitting the dataset into training and test set.
14. from sklearn.model\_selection **import** train\_test\_split
15. x\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size= 0.25, random\_state=0)
- 16.
17. #feature Scaling
18. from sklearn.preprocessing **import** StandardScaler

19. st\_x= StandardScaler()

20. x\_train= st\_x.fit\_transform(x\_train)    21. x\_test= st\_x.transform(x\_test)

In the above code, we have pre-processed the data. Where we have loaded the dataset, which is given as:



Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0
14	15628972	Male	18	82000	0
15	15697686	Male	29	80000	0

## 2. Fitting a Decision-Tree algorithm to the Training set

Now we will fit the model to the training set. For this, we will import the **DecisionTreeClassifier** class from **sklearn.tree** library. Below is the code for it:

1. #Fitting Decision Tree classifier to the training set
2. From sklearn.tree **import** DecisionTreeClassifier
3. classifier= DecisionTreeClassifier(criterion='entropy', random\_state=0)
4. classifier.fit(x\_train, y\_train)

In the above code, we have created a classifier object, in which we have passed two main parameters;

- **"criterion='entropy'":** Criterion is used to measure the quality of split, which is calculated by information gain given by entropy.
- **random\_state=0":** For generating the random states.

Below is the output for this:

```
Out[8]:
DecisionTreeClassifier(class_weight=None, criterion='entropy',
max_depth=None, max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort=False,
random_state=0, splitter='best')
```

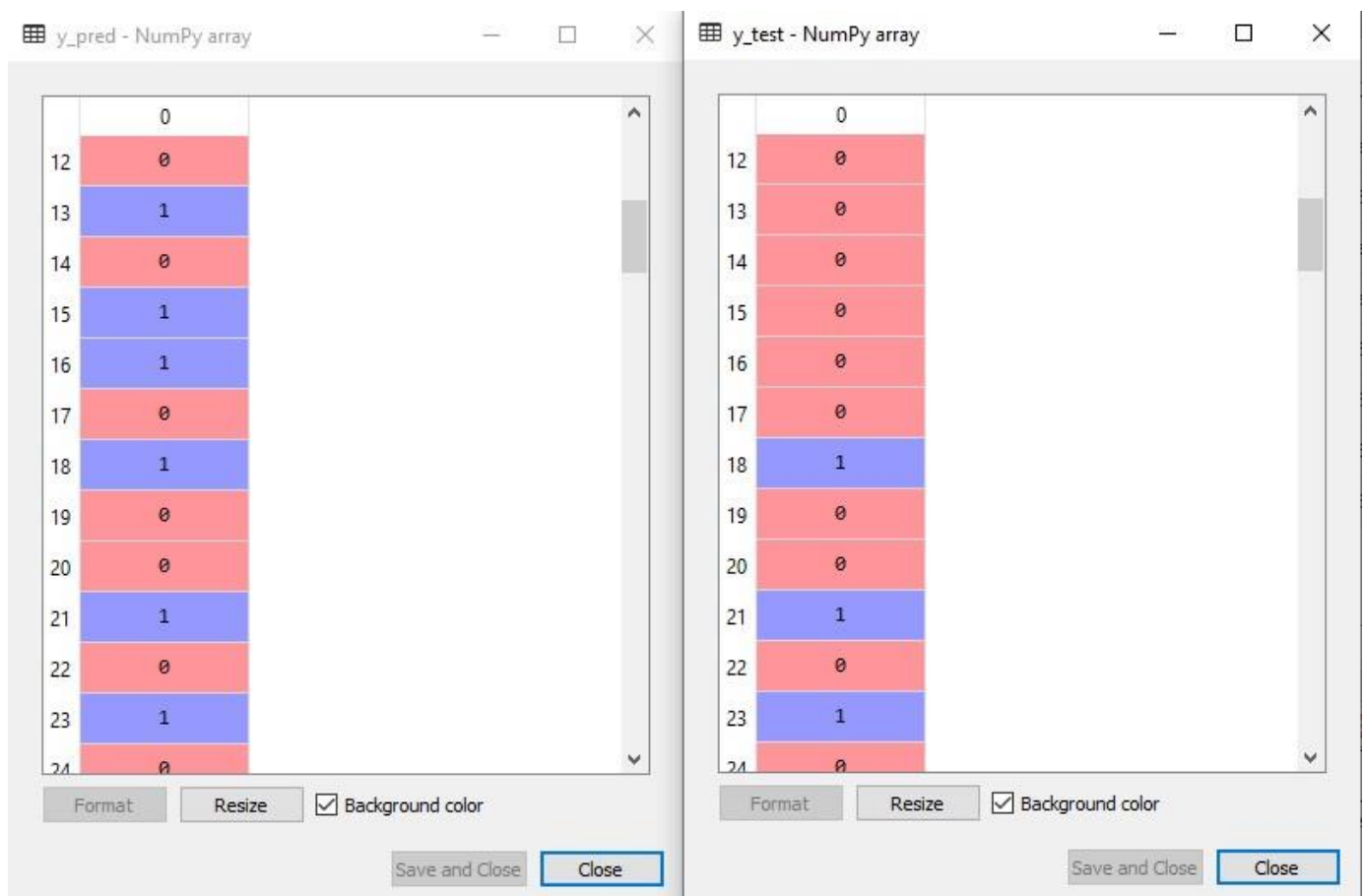
### 3. Predicting the test result

Now we will predict the test set result. We will create a new prediction vector **y\_pred**. Below is the code for it:

1. #Predicting the test set result
2. `y_pred= classifier.predict(x_test)`

**Output:**

In the below output image, the predicted output and real test output are given. We can clearly see that there are some values in the prediction vector, which are different from the real vector values. These are prediction errors.

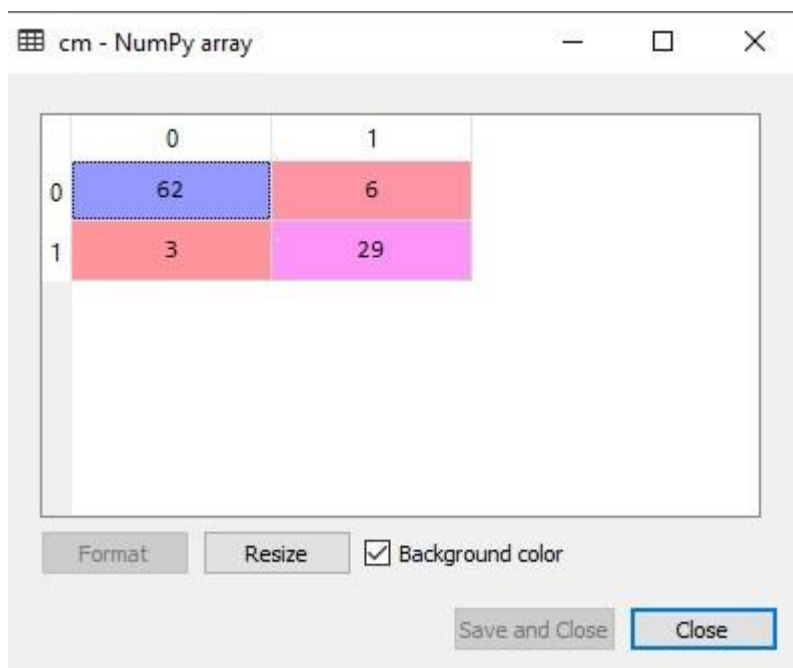


## 4. Test accuracy of the result (Creation of Confusion matrix)

In the above output, we have seen that there were some incorrect predictions, so if we want to know the number of correct and incorrect predictions, we need to use the confusion matrix. Below is the code for it:

1. #Creating the Confusion matrix
2. from sklearn.metrics **import** confusion\_matrix
3. cm= confusion\_matrix(y\_test, y\_pred)

**Output:**



In the above output image, we can see the confusion matrix, which has **6+3= 9 incorrect predictions** and **62+29=91 correct predictions**. Therefore, we can say that compared to other classification models, the Decision Tree classifier made a good prediction.

## 5. Visualizing the training set result:

Here we will visualize the training set result. To visualize the training set result we will plot a graph for the decision tree classifier. The classifier will predict yes or No for the users who have either Purchased or Not purchased the SUV car as we did in [Logistic Regression](#). Below is the code for it:

1. #Visulaizing the trianing set result
2. from matplotlib.colors **import** ListedColormap
3. x\_set, y\_set = x\_train, y\_train
4. x1, x2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step = 0.01),
5. nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01))
6. mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
7. alpha = 0.75, cmap = ListedColormap(('purple','green' )))

```

8.     mtp.xlim(x1.min(), x1.max())
9.     mtp.ylim(x2.min(), x2.max())
10.    for i, j in enumerate(nm.unique(y_set)):
11.        mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
12.                    c = ListedColormap(('purple', 'green'))(i), label = j)
13.    mtp.title('Decision Tree Algorithm (Training set)')
14.    mtp.xlabel('Age')
15.    mtp.ylabel('Estimated Salary')
16.    mtp.legend()
17.    mtp.show()

```

**Output:**



The above output is completely different from the rest classification models. It has both vertical and horizontal lines that are splitting the dataset according to the age and estimated salary variable.

As we can see, the tree is trying to capture each dataset, which is the case of overfitting.

## 6. Visualizing the test set result:

Visualization of test set result will be similar to the visualization of the training set except that the training set will be replaced with the test set.

```

1.     #Visulaizing the test set result
2.     from matplotlib.colors import ListedColormap
3.     x_set, y_set = x_test, y_test
4.     x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max()
5.                             + 1, step = 0.01),
6.                           nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
7.     mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
8.                 alpha = 0.75, cmap = ListedColormap(('purple','green' )))

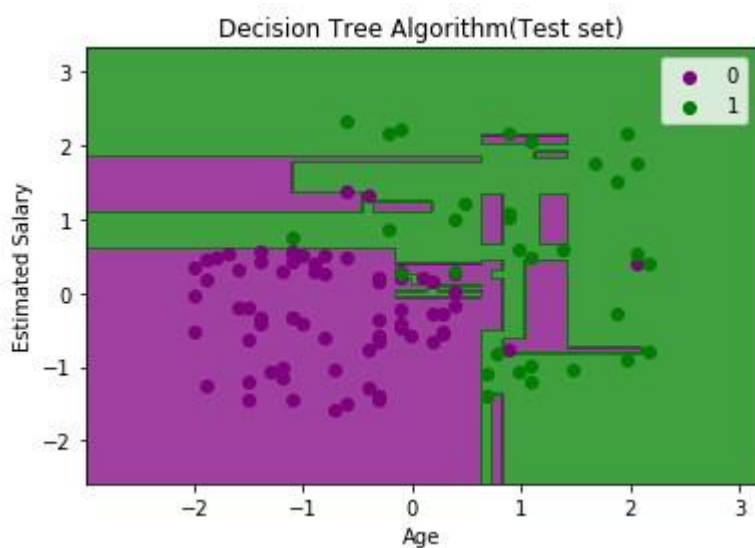
```

```

8.     mtp.xlim(x1.min(), x1.max())
9.     mtp.ylim(x2.min(), x2.max())
10.    for i, j in enumerate(nm.unique(y_set)):
11.        mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
12.                    c = ListedColormap(('purple', 'green'))(i), label = j)
13.    mtp.title('Decision Tree Algorithm(Test set)')
14.    mtp.xlabel('Age')
15.    mtp.ylabel('Estimated Salary')
16.    mtp.legend()
17.    mtp.show()

```

**Output:**



As we can see in the above image that there are some green data points within the purple region and vice versa. So, these are the incorrect predictions which we have discussed in the confusion matrix.

## Random Forest Algorithm

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of **ensemble learning**, which is a process of *combining multiple classifiers to solve a complex problem and to improve the performance of the model*.

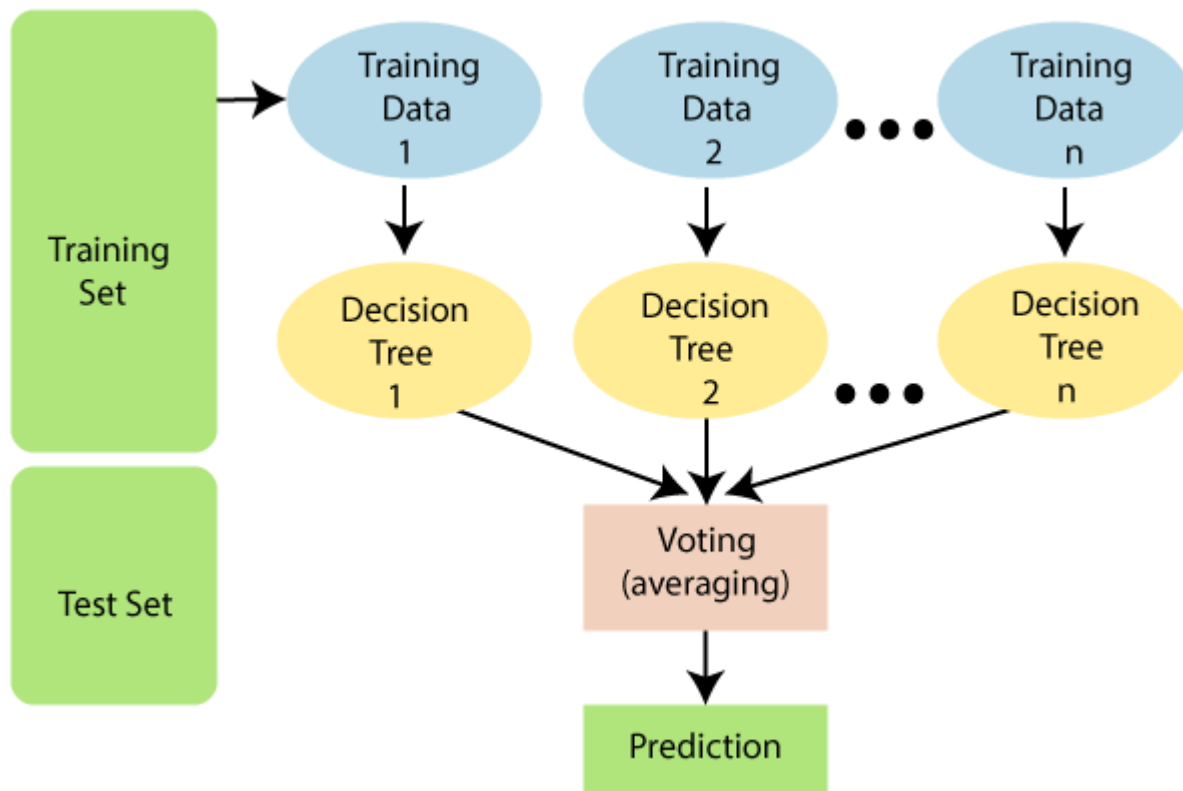
As the name suggests, *"Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy"*



of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

**The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.**

The below diagram explains the working of the Random Forest algorithm.



*Note: To better understand the Random Forest Algorithm, you should have knowledge of the Decision Tree Algorithm.*

## Assumptions for Random Forest

Since the random forest combines multiple trees to predict the class of the dataset, it is possible that some decision trees may predict the correct output, while others may not. But together, all the trees predict the correct output. Therefore, below are two assumptions for a better Random forest classifier:

- There should be some actual values in the feature variable of the dataset so that the classifier can predict accurate results rather than a guessed result.
- The predictions from each tree must have very low correlations.

## Why use Random Forest?

Below are some points that explain why we should use the Random Forest algorithm:

- It takes less training time as compared to other algorithms.
- It predicts output with high accuracy, even for the large dataset it runs efficiently.
- It can also maintain accuracy when a large proportion of data is missing.

# How does Random Forest algorithm work?

Random Forest works in two-phase first is to create the random forest by combining N decision tree, and second is to make predictions for each tree created in the first phase.

The Working process can be explained in the below steps and diagram:

**Step-1:** Select random K data points from the training set.

**Step-2:** Build the decision trees associated with the selected data points (Subsets).

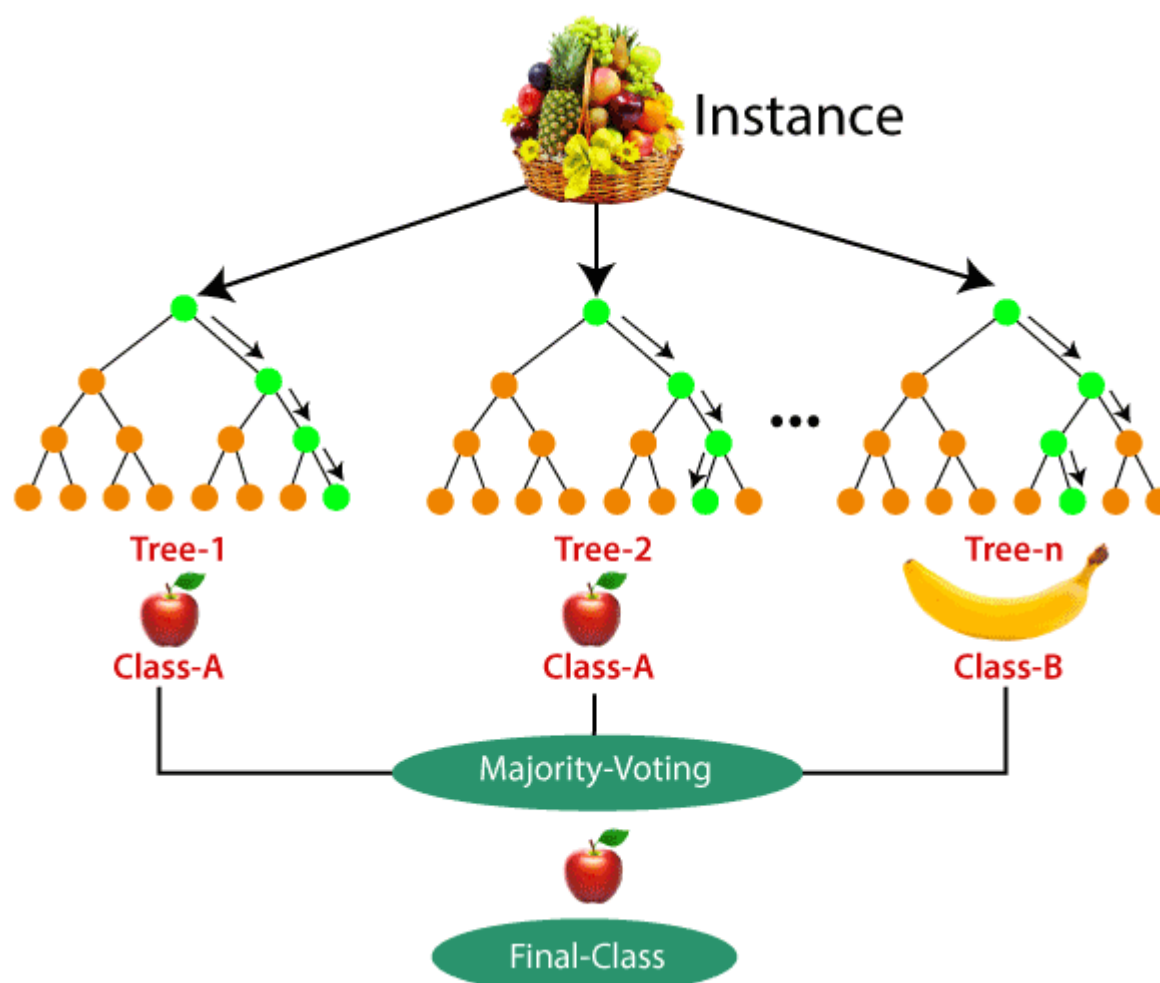
**Step-3:** Choose the number N for decision trees that you want to build.

**Step-4:** Repeat Step 1 & 2.

**Step-5:** For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

The working of the algorithm can be better understood by the below example:

**Example:** Suppose there is a dataset that contains multiple fruit images. So, this dataset is given to the Random forest classifier. The dataset is divided into subsets and given to each decision tree. During the training phase, each decision tree produces a prediction result, and when a new data point occurs, then based on the majority of results, the Random Forest classifier predicts the final decision. Consider the below image:



# Applications of Random Forest

There are mainly four sectors where Random forest mostly used:

1. **Banking:** Banking sector mostly uses this algorithm for the identification of loan risk.
2. **Medicine:** With the help of this algorithm, disease trends and risks of the disease can be identified.
3. **Land Use:** We can identify the areas of similar land use by this algorithm.
4. **Marketing:** Marketing trends can be identified using this algorithm.

## Advantages of Random Forest

- Random Forest is capable of performing both Classification and Regression tasks.
- It is capable of handling large datasets with high dimensionality.
- It enhances the accuracy of the model and prevents the overfitting issue.

## Disadvantages of Random Forest

- Although random forest can be used for both classification and regression tasks, it is not more suitable for Regression tasks.

## Python Implementation of Random Forest Algorithm

Now we will implement the Random Forest Algorithm tree using Python. For this, we will use the same dataset "user\_data.csv", which we have used in previous classification models. By using the same dataset, we can compare the Random Forest classifier with other classification models such as [Decision tree Classifier](#), [KNN](#), [SVM](#), [Logistic Regression](#), etc.

Implementation Steps are given below:

- Data Pre-processing step
- Fitting the Random forest algorithm to the Training set
- Predicting the test result
- Test accuracy of the result (Creation of Confusion matrix)
- Visualizing the test set result.

### 1.Data Pre-Processing Step:

Below is the code for the pre-processing step:

1. # importing libraries

```

2. import numpy as nm
3. import matplotlib.pyplot as mtp  4. import pandas as pd
5.
6. #importing datasets
7. data_set= pd.read_csv('user_data.csv')
8.
9. #Extracting Independent and dependent Variable
10. x= data_set.iloc[:, [2,3]].values    11. y= data_set.iloc[:,
    4].values
12.
13. # Splitting the dataset into training and test set.
14. from sklearn.model_selection import train_test_split
15. x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
16.
17. #feature Scaling
18. from sklearn.preprocessing import StandardScaler
19. st_x= StandardScaler()
20. x_train= st_x.fit_transform(x_train)    21. x_test= st_x.transform(x_test)

```

In the above code, we have pre-processed the data. Where we have loaded the dataset, which is given as:

Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0

Format    Resize    ☒ Background color    ☐ Column min/max    Save and Close    Close

## 2. Fitting the Random Forest algorithm to the training set:

Now we will fit the Random forest algorithm to the training set. To fit it, we will import the **RandomForestClassifier** class from the **sklearn.ensemble** library. The code is given below:

1. #Fitting Decision Tree classifier to the training set
2. from sklearn.ensemble **import** RandomForestClassifier
3. classifier= RandomForestClassifier(n\_estimators= 10, criterion="entropy")
4. classifier.fit(x\_train, y\_train)

In the above code, the classifier object takes below parameters:

- **n\_estimators**= The required number of trees in the Random Forest. The default value is 10. We can choose any number but need to take care of the overfitting issue.
- **criterion**= It is a function to analyze the accuracy of the split. Here we have taken "entropy" for the information gain.

**Output:**

```
RandomForestClassifier(bootstrap=True, class_weight=None,
criterion='entropy',
max_depth=None,
max_features='auto', max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
```

```
min_weight_fraction_leaf=0.0, n_estimators=10,  
n_jobs=None, oob_score=False, random_state=None,  
verbose=0, warm_start=False)
```

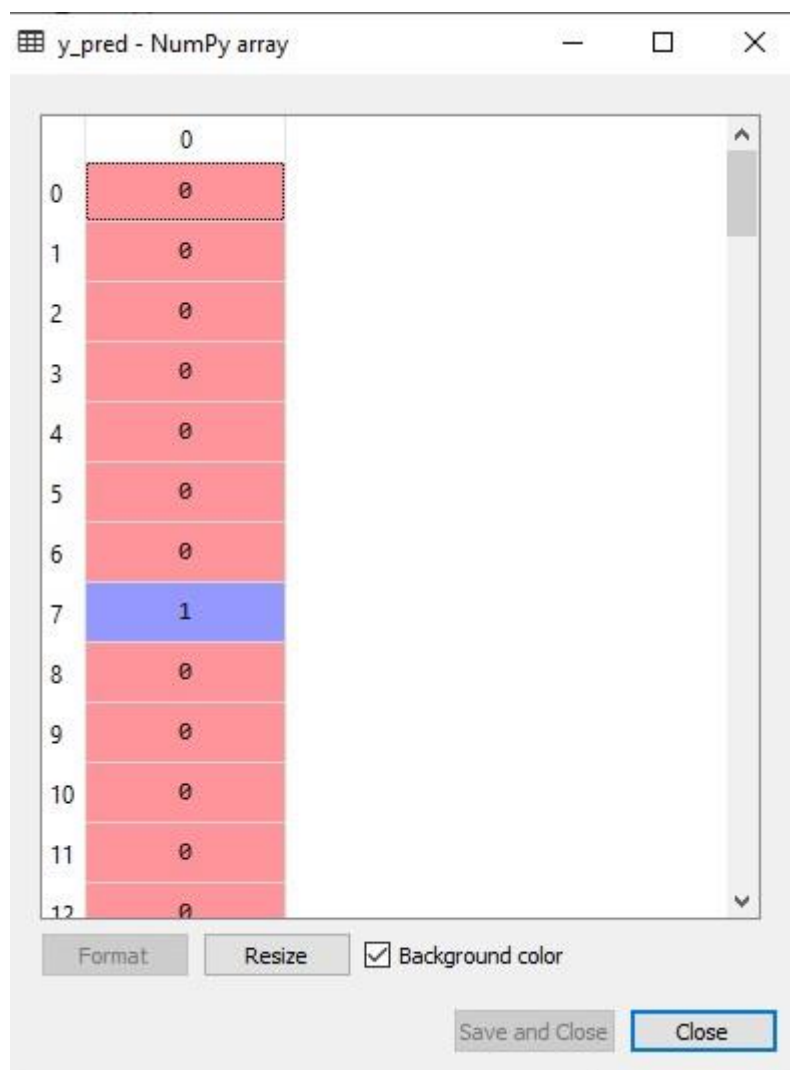
### 3. Predicting the Test Set result

Since our model is fitted to the training set, so now we can predict the test result. For prediction, we will create a new prediction vector `y_pred`. Below is the code for it:

1. `#Predicting the test set result`
2. `y_pred= classifier.predict(x_test)`

#### Output:

The prediction vector is given as:



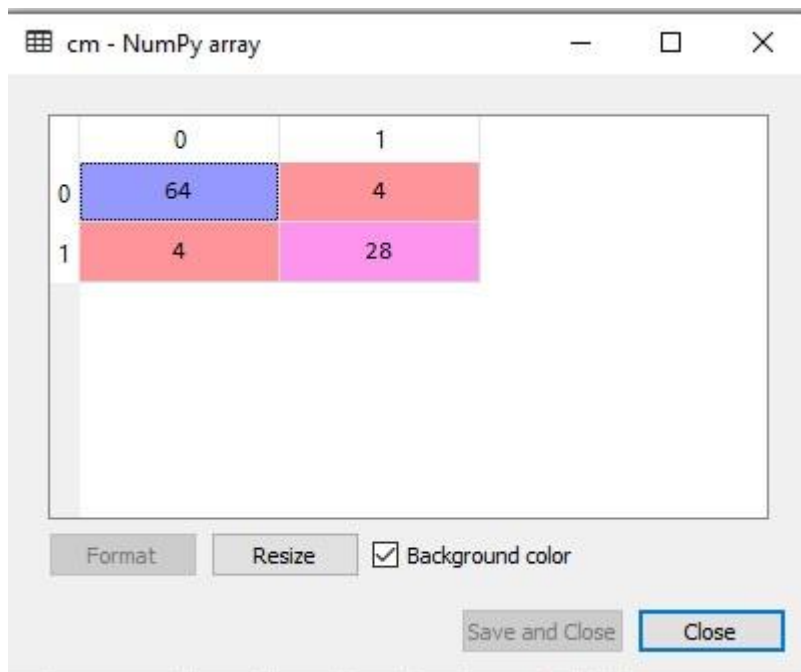
By checking the above prediction vector and test set real vector, we can determine the incorrect predictions done by the classifier.

### 4. Creating the Confusion Matrix

Now we will create the confusion matrix to determine the correct and incorrect predictions. Below is the code for it:

1. #Creating the Confusion matrix
2. from sklearn.metrics **import** confusion\_matrix
3. cm= confusion\_matrix(y\_test, y\_pred)

**Output:**



As we can see in the above matrix, there are **4+4= 8 incorrect predictions** and **64+28= 92 correct predictions**.

## 5. Visualizing the training Set result

Here we will visualize the training set result. To visualize the training set result we will plot a graph for the Random forest classifier. The classifier will predict yes or No for the users who have either Purchased or Not purchased the SUV car as we did in [Logistic Regression](#). Below is the code for it:

1. from matplotlib.colors **import** ListedColormap
2. x\_set, y\_set = x\_train, y\_train
3. x1, x2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step = 0.01),
4. nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01))
5. mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),

```

6.     alpha = 0.75, cmap = ListedColormap(('purple','green' )))
7.     mtp.xlim(x1.min(), x1.max())
8.     mtp.ylim(x2.min(), x2.max())
9.     for i, j in enumerate(nm.unique(y_set)):
10.    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
11.    c = ListedColormap(('purple', 'green'))(i), label = j)
12.    mtp.title('Random Forest Algorithm (Training set)')
13.    mtp.xlabel('Age')
14.    mtp.ylabel('Estimated Salary')
15.    mtp.legend()
16.    mtp.show()

```

**Output:**



The above image is the visualization result for the Random Forest classifier working with the training set result. It is very much similar to the Decision tree classifier. Each data point corresponds to each user of the user\_data, and the purple and green regions are the prediction regions. The purple region is classified for the users who did not purchase the SUV car, and the green region is for the users who purchased the SUV.

So, in the Random Forest classifier, we have taken 10 trees that have predicted Yes or NO for the Purchased variable. The classifier took the majority of the predictions and provided the result.

## 6. Visualizing the test set result

Now we will visualize the test set result. Below is the code for it:

```

1.     #Visulaizing the test set result
2.     from matplotlib.colors import ListedColormap
3.     x_set, y_set = x_test, y_test
4.     x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max()
5.     + 1, step = 0.01),
6.     nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))

```

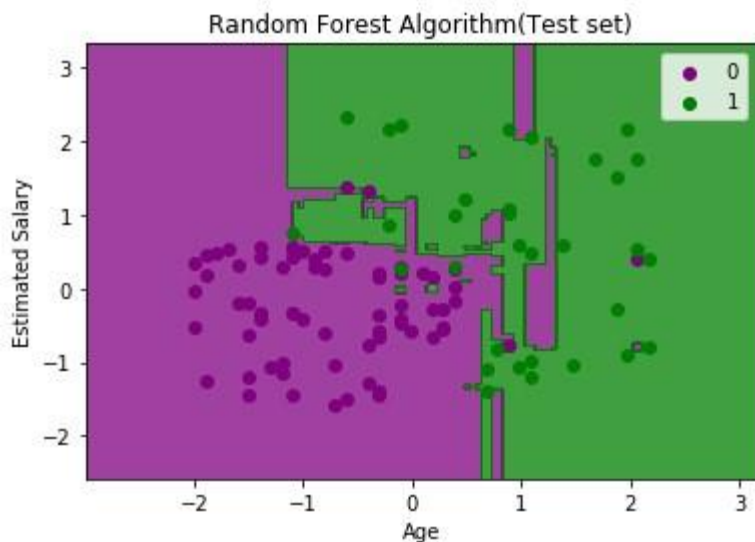


```

6.     mtp.contourf(x1,                x2,                classifier.predict(nm.array([x1.ravel(),
x2.ravel()])).T).reshape(x1.shape),
7.     alpha = 0.75, cmap = ListedColormap(('purple', 'green' )))
8.     mtp.xlim(x1.min(), x1.max())
9.     mtp.ylim(x2.min(), x2.max())
10.    for i, j in enumerate(nm.unique(y_set)):
11.        mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
12.        c = ListedColormap(('purple', 'green'))(i), label = j)
13.    mtp.title('Random Forest Algorithm(Test set)')
14.    mtp.xlabel('Age')
15.    mtp.ylabel('Estimated Salary')
16.    mtp.legend()
17.    mtp.show()

```

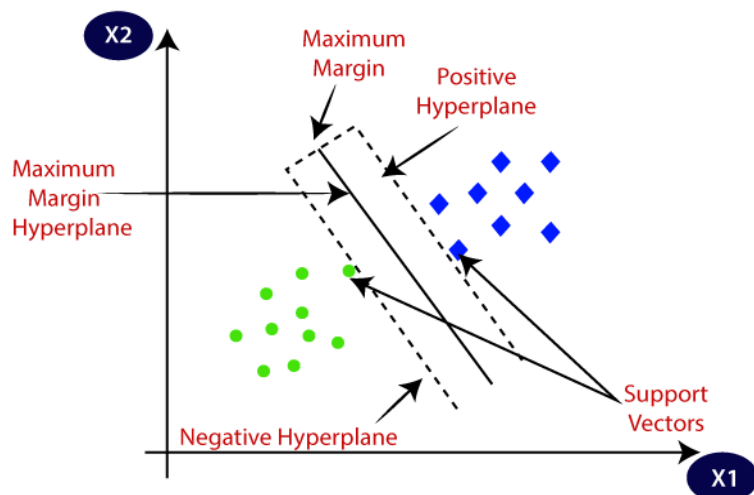
### Output:



The above image is the visualization result for the test set. We can check that there is a minimum number of incorrect predictions (8) without the Overfitting issue. We will get different results by changing the number of trees in the classifier.

# Support Vector Regression (SVR):

- Support vector regression uses the same principle of Support Vector Machine (SVM) but for regression problems.
- SVM is a supervised learning algorithm
- The goal of the SVM algorithm is to create the best line or decision boundary so that we can easily put the new data point in the correct category in the future.
- This best line or 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.
- **Hyperplane:** There can be multiple lines/decision boundaries to segregate the classes in ndimensional space, but we need to find out the best decision boundary that helps to classify the data points. We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

## ○ 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.

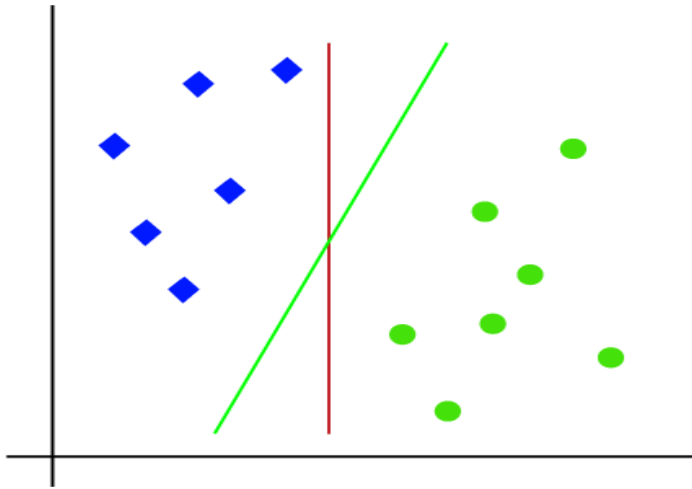
□ **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as nonlinear data and classifier used is called as Non-linear SVM classifier.

When we are taking about non linear data we make use of kernal function which is used to covert low dimensionality problem to high dimensional one.

z

□ **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  $x_1$  and  $x_2$ . We want a classifier that can classify the pair( $x_1$ ,  $x_2$ ) of coordinates in either green or blue. 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**.
- If we observe both the red and green lines are hyperplanes but if we observe closely. The green line will differentiate two classes better than red line as it is having maximum margin.

□

## Linear SVM

**example:**

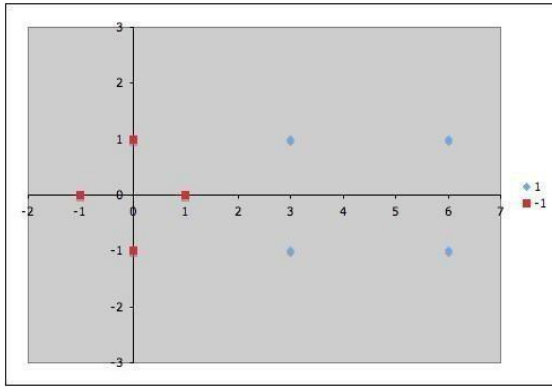
Suppose we are given the following positively labeled data points,

$$\left\{ \begin{pmatrix} 3 \\ 1 \end{pmatrix}, \begin{pmatrix} 3 \\ -1 \end{pmatrix}, \begin{pmatrix} 6 \\ 1 \end{pmatrix}, \begin{pmatrix} 6 \\ -1 \end{pmatrix} \right\}$$

and the following negatively labeled data points,

$$\left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \end{pmatrix} \right\}$$

- Plot the data points on a graph



- By inspection, it should be obvious that there are **three** support vectors,

$$\left\{ s_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, s_2 = \begin{pmatrix} 3 \\ 1 \end{pmatrix}, s_3 = \begin{pmatrix} 3 \\ -1 \end{pmatrix} \right\}$$

- Each vector is augmented with a 1 as a bias input

- So,  $s_1 = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$ , then  $\tilde{s}_1 = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$

- Similarly,

- $s_2 = \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix}$ , then  $\tilde{s}_2 = \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix}$  and  $s_3 = \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix}$ , then  $\tilde{s}_3 = \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix}$

- As there are three support vectors, we need to calculate 3 variables  $\alpha_1, \alpha_2, \alpha_3$  for calculating the base vector.

$$\begin{aligned} \alpha_1 \tilde{s}_1 \cdot \tilde{s}_1 + \alpha_2 \tilde{s}_2 \cdot \tilde{s}_1 + \alpha_3 \tilde{s}_3 \cdot \tilde{s}_1 &= -1 & \alpha_1(1+0+1) + \alpha_2(3+0+1) + \alpha_3(3+0+1) &= -1 \\ \alpha_1 \tilde{s}_1 \cdot \tilde{s}_2 + \alpha_2 \tilde{s}_2 \cdot \tilde{s}_2 + \alpha_3 \tilde{s}_3 \cdot \tilde{s}_2 &= +1 & \alpha_1(3+0+1) + \alpha_2(9+1+1) + \alpha_3(9-1+1) &= 1 \\ \alpha_1 \tilde{s}_1 \cdot \tilde{s}_3 + \alpha_2 \tilde{s}_2 \cdot \tilde{s}_3 + \alpha_3 \tilde{s}_3 \cdot \tilde{s}_3 &= +1 & \alpha_1(3+0+1) + \alpha_2(9-1+1) + \alpha_3(9+1+1) &= 1 \end{aligned}$$

$$\begin{aligned} \alpha_1 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + \alpha_3 \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} &= -1 & 2\alpha_1 + 4\alpha_2 + 4\alpha_3 &= -1 \\ \alpha_1 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} + \alpha_3 \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix} \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} &= 1 & 4\alpha_1 + 11\alpha_2 + 9\alpha_3 &= 1 \\ \alpha_1 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix} + \alpha_3 \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix} \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix} &= 1 & 4\alpha_1 + 9\alpha_2 + 11\alpha_3 &= 1 \end{aligned}$$

$$\begin{aligned} \alpha_1 &= -3.5 \\ \alpha_2 &= 0.75 \\ \alpha_3 &= 0.75 \end{aligned}$$

$$\begin{aligned} \tilde{w} &= \sum_i \alpha_i \tilde{s}_i \\ &= -3.5 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + 0.75 \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} + 0.75 \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} 1 \\ 0 \\ -2 \end{pmatrix} \end{aligned}$$

- Finally, remembering that our vectors are augmented with a bias.
- We can equate the last entry in  $\tilde{w}$  as the hyperplane offset  $b$  and write the separating
- Hyperplane equation  $\mathbf{y} = \mathbf{w}\mathbf{x} + \mathbf{b}$
- with  $\mathbf{w} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$  and  $\mathbf{b} = -2$ .

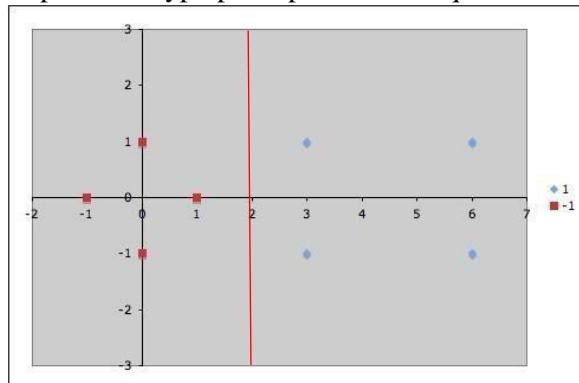
By calculating this we can draw a hyperplane

Here (1,0) represents Hyperplane is parallel to Y-axis.

If it is (0,1) Hyperplane will be parallel to X-axis

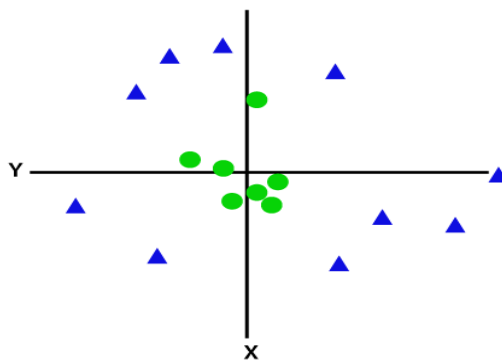
If it is (1,1) it will be 45 degrees with respect to X and Y axis.

As the bias value is -2 It represents Hyperplane present in 1<sup>st</sup> quadrant and touches 2



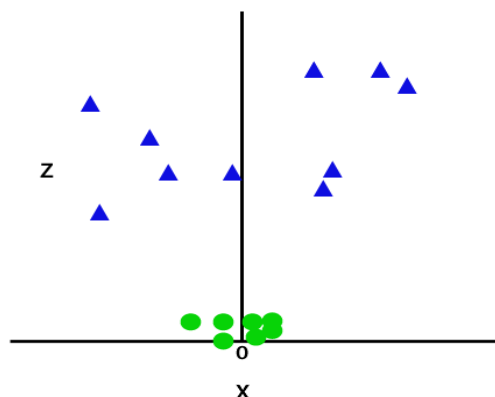
### Non-Linear SVM:

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

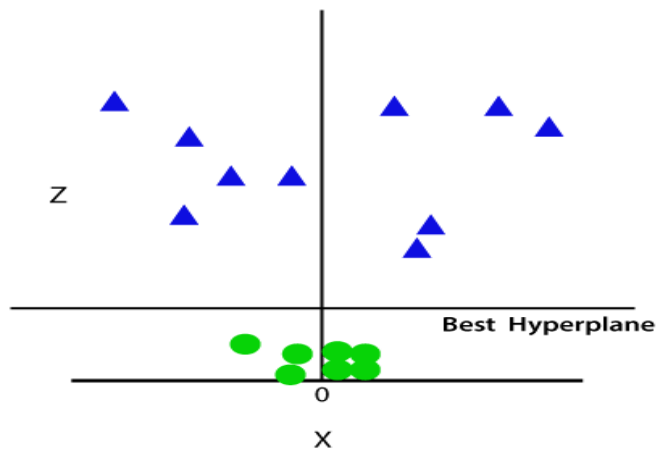


So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:  $z = x^2 + y^2$

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



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



Non Linear SVM example

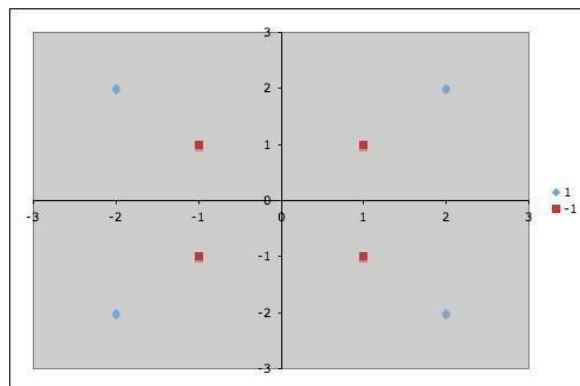
Suppose we are given the following positively labeled data points,

$$\left\{ \begin{pmatrix} 2 \\ 2 \end{pmatrix}, \begin{pmatrix} -2 \\ -2 \end{pmatrix}, \begin{pmatrix} -2 \\ -2 \end{pmatrix}, \begin{pmatrix} -2 \\ 2 \end{pmatrix} \right\}$$

and the following negatively labeled data points,

$$\left\{ \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \end{pmatrix}, \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \begin{pmatrix} -1 \\ 1 \end{pmatrix} \right\}$$

- Plot the data points on a graph



- Our goal, again, is to discover a separating hyperplane that accurately discriminates the two classes.
- Of course, it is obvious that no such hyperplane exists in the input space
- Therefore, we must use a nonlinear SVM (that is, we need to convert data from one feature space to another).

$$\Phi_1 \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{cases} \begin{pmatrix} 4 - x_2 + |x_1 - x_2| \\ 4 - x_1 + |x_1 - x_2| \\ x_1 \\ x_2 \end{pmatrix} & \text{if } \sqrt{x_1^2 + x_2^2} > 2 \\ \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} & \text{otherwise} \end{cases}$$

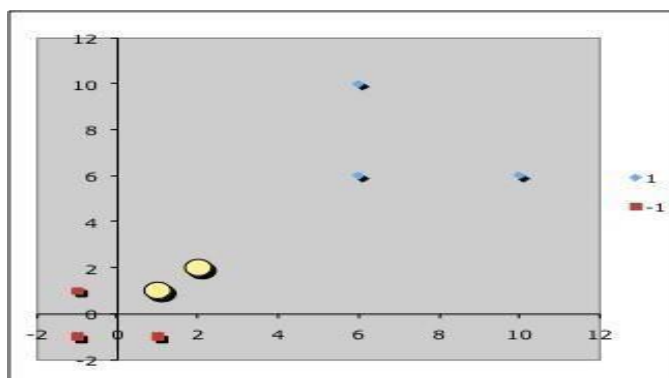
### Positive Examples

$$\left\{ \begin{pmatrix} 2 \\ 2 \end{pmatrix}, \begin{pmatrix} 2 \\ -2 \end{pmatrix}, \begin{pmatrix} -2 \\ -2 \end{pmatrix}, \begin{pmatrix} -2 \\ 2 \end{pmatrix} \right\} \rightarrow \left\{ \begin{pmatrix} 2 \\ 2 \end{pmatrix}, \begin{pmatrix} 10 \\ 6 \end{pmatrix}, \begin{pmatrix} 6 \\ 6 \end{pmatrix}, \begin{pmatrix} 6 \\ 10 \end{pmatrix} \right\}$$

### Negative Examples

$$\left\{ \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \end{pmatrix}, \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \begin{pmatrix} -1 \\ 1 \end{pmatrix} \right\} \rightarrow \left\{ \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \end{pmatrix}, \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \begin{pmatrix} -1 \\ 1 \end{pmatrix} \right\}$$

Now plot the new data points in graph



- Now we can easily identify the support vectors,

$$\left\{ s_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, s_2 = \begin{pmatrix} 2 \\ 2 \end{pmatrix} \right\}$$

- Each vector is augmented with a 1 as a bias input

$$\tilde{s}_1 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \tilde{s}_2 = \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix}$$

- As there are two support vectors, we need to calculate 2 variables alpha 1,2 for calculating the base vector.

$$\begin{aligned} \alpha_1 \tilde{s}_1 \cdot \tilde{s}_1 + \alpha_2 \tilde{s}_2 \cdot \tilde{s}_1 &= -1 & \alpha_1(1+1+1) + \alpha_2(2+2+1) &= -1 \\ \alpha_1 \tilde{s}_1 \cdot \tilde{s}_2 + \alpha_2 \tilde{s}_2 \cdot \tilde{s}_2 &= +1 & \alpha_1(2+2+1) + \alpha_2(4+4+1) &= 1 \end{aligned}$$

$$\alpha_1 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = -1$$

$$\begin{aligned} 3\alpha_1 + 5\alpha_2 &= -1 \\ 5\alpha_1 + 9\alpha_2 &= 1 \end{aligned}$$

$$\alpha_1 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix} = 1$$

$$\begin{aligned} \alpha_1 &= -7 \\ \alpha_2 &= 4 \end{aligned}$$

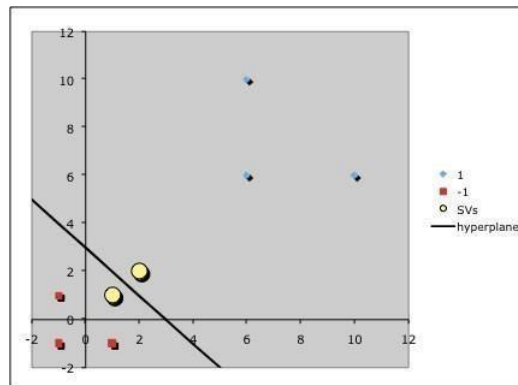
$$\begin{aligned}\tilde{w} &= \sum_i \alpha_i \tilde{s}_i = -7 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + 4 \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} 1 \\ 1 \\ -3 \end{pmatrix}\end{aligned}$$

- Finally, remembering that our vectors are augmented with a bias.
- We can equate the last entry in  $\tilde{w}$  as the hyperplane offset  $b$  and write the separating
- Hyperplane equation  $y = wx + b$
- with  $w = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$  and  $b = -3$ .

By calculating this we can draw a hyperplane

Here (1,1) represents Hyperplane will be 45 degrees with respective X and Y axis.

As the bias value is -3 It represents Hyperplane present in 1<sup>st</sup> quadrant and makes 45 degrees



- SVR is similar to linear regression in that the equation of line is  $y = WX + b$  in SVR that straight line is called hyperplane and like other regression models that try to minimise the error between real and predicted values the SVM tries to fit the best line with an threshold value **Advantages:**
  - Gives good results even if there is not enough information about the data also works well with unstructured data
  - Solves complex problems with a convenient kernel solution function
  - Relatively good scaling of high dimensional data
- Disadvantages:**
  - It is difficult to choose the appropriate kernel solution function
  - Training time is long when using large data sets
  - It may be difficult to interpret and understand because of problems caused by personal factors and the weights of variables



## Boosting

→ Boosting is an ensemble modeling technique

↓

We consider multiple other models in prediction process

→ Boosting attempts to build a strong classifier from number of weak classifiers

→ Boosting is done by building a model by using weak model in series

→ Firstly a model is built from training data

→ Then the second model is built from the training data which tries to correct the errors present in first model.

→ This procedure is continued and models are added until either the complete training data set is predicted correctly or max number of models are added.

→ AdaBoost was the first really successful boosting algorithm developed for the purpose of Binary classification. AdaBoost is <sup>short</sup> for Adaptive Boosting.

### Algorithm

1. Initialise the dataset and assign equal weight to each of the data point.
2. Provide this as input to the model and identify the wrongly classified data points.
3. Increase the weight of the wrongly classified data points.

2023/4/7



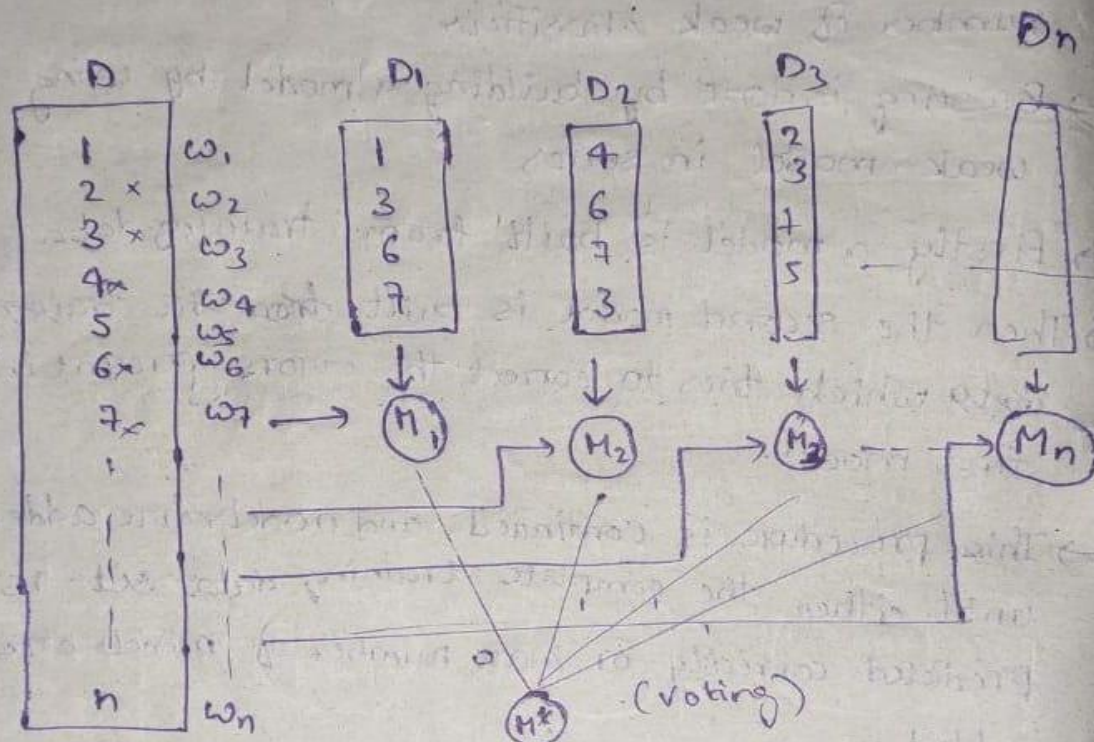
4. If (got required results)

Go to step 5

else

Go to step 2

5. End.



→ At start all the weights are same that means when we create our first dataset  $D_1$  which is subset of  $D$  through which we are training our first model

→ From  $D$  we randomly select some instances for  $D_1$  they have same probability

→ On the basis of  $D_1$  we train  $M_1$  model  
↓  
training dataset

→ After that we give all instances of  $D$  are given



as input to our model  $M_1$ , for testing/classifying.

→ Our  $M_1$  will predict and classify whether the instance belong to class A or class B.

→ There is no guarantee that  $M_1$  will predict correctly.

There may be some misclassification.

→ If 4, 6 & 7 are wrongly predicted then we need to update the weights of those.

→ Again we need to create second data set  $D_2$  & continue the process.

→ We need to focus on the weights which were wrongly predicted by previous model.

→ Increasing the weights results in increasing the probability to get selected while creating new datasets.

→ Like this we create  $n$  models from that we create one Model  $M^*$ .

→ When the training dataset is given to all models.

$M^*$  will conduct voting and predict the class label.

→ Here one strong model is build from many weak models.