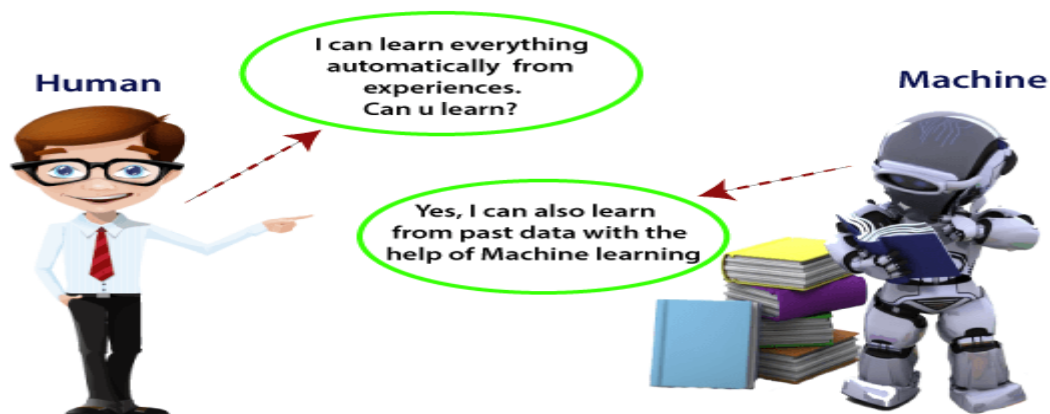


UNIT-01

What is Machine Learning

In the real world, we are surrounded by humans who can learn everything from their experiences with their learning capability, and we have computers or machines which work on our instructions. But can a machine also learn from experiences or past data like a human does? So here comes the role of **Machine Learning**.



Machine Learning is said as a subset of **artificial intelligence** that is mainly concerned with the development of algorithms which allow a computer to learn from the data and past experiences on their own. The term machine learning was first introduced by **Arthur Samuel** in **1959**. We can define it in a summarized way as:

Machine learning enables a machine to automatically learn from data, improve performance from experiences, and predict things without being explicitly programmed.

With the help of sample historical data, which is known as **training data**, machine learning algorithms build a **mathematical model** that helps in making predictions or decisions without being explicitly programmed. Machine learning brings computer science and statistics together for creating predictive models. Machine learning constructs or uses the algorithms that learn from historical data. The more we will provide the information, the higher will be the performance.

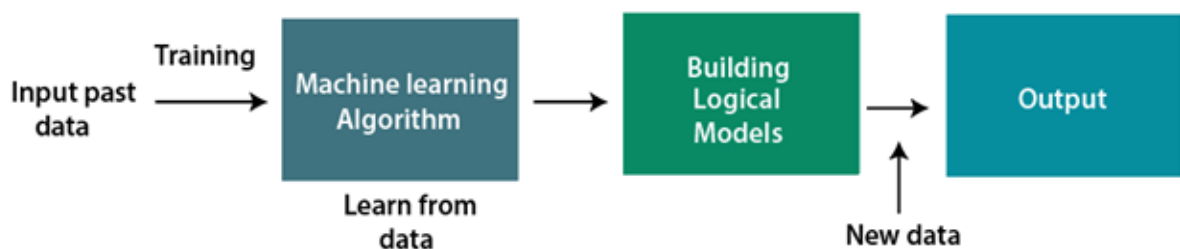
A machine has the ability to learn if it can improve its performance by gaining more data.

How does Machine Learning work

A Machine Learning system **learns from historical data, builds the prediction models, and whenever it receives new data, predicts the output for it.** The

accuracy of predicted output depends upon the amount of data, as the huge amount of data helps to build a better model which predicts the output more accurately.

Suppose we have a complex problem, where we need to perform some predictions, so instead of writing a code for it, we just need to feed the data to generic algorithms, and with the help of these algorithms, machine builds the logic as per the data and predict the output. Machine learning has changed our way of thinking about the problem. The below block diagram explains the working of Machine Learning algorithm:



Features of Machine Learning:

- Machine learning uses data to detect various patterns in a given dataset.
- It can learn from past data and improve automatically.
- It is a data-driven technology.
- Machine learning is much similar to data mining as it also deals with the huge amount of the data.

Need for Machine Learning

The need for machine learning is increasing day by day. The reason behind the need for machine learning is that it is capable of doing tasks that are too complex for a person to implement directly. As a human, we have some limitations as we cannot access the huge amount of data manually, so for this, we need some computer systems and here comes the machine learning to make things easy for us.

We can train machine learning algorithms by providing them the huge amount of data and let them explore the data, construct the models, and predict the required output automatically. The performance of the machine learning algorithm depends on the amount of data, and it can be determined by the cost function. With the help of machine learning, we can save both time and money.

The importance of machine learning can be easily understood by its uses cases, Currently, machine learning is used in **self-driving cars, cyber fraud detection, face**

recognition, and **friend suggestion by Facebook**, etc. Various top companies such as Netflix and Amazon have build machine learning models that are using a vast amount of data to analyze the user interest and recommend product accordingly.

Following are some key points which show the importance of Machine Learning:

- Rapid increment in the production of data
- Solving complex problems, which are difficult for a human
- Decision making in various sector including finance
- Finding hidden patterns and extracting useful information from data.

Classification of Machine Learning

At a broad level, machine learning can be classified into three types:

1. **Supervised learning**
2. **Unsupervised learning**
3. **Reinforcement learning**



1) Supervised Learning

Supervised learning is a type of machine learning method in which we provide sample labeled data to the machine learning system in order to train it, and on that basis, it predicts the output.

The system creates a model using labeled data to understand the datasets and learn about each data, once the training and processing are done then we test the model by providing a sample data to check whether it is predicting the exact output or not.

The goal of supervised learning is to map input data with the output data. The supervised learning is based on supervision, and it is the same as when a student learns things in the supervision of the teacher. The example of supervised learning is **spam filtering**.

Supervised learning can be grouped further in two categories of algorithms:

- **Classification**
- **Regression**

2) Unsupervised Learning

Unsupervised learning is a learning method in which a machine learns without any supervision.

The training is provided to the machine with the set of data that has not been labeled, classified, or categorized, and the algorithm needs to act on that data without any supervision. The goal of unsupervised learning is to restructure the input data into new features or a group of objects with similar patterns.

In unsupervised learning, we don't have a predetermined result. The machine tries to find useful insights from the huge amount of data. It can be further classified into two categories of algorithms:

- **Clustering**
- **Association**

3) Reinforcement Learning

Reinforcement learning is a feedback-based learning method, in which a learning agent gets a reward for each right action and gets a penalty for each wrong action. The agent learns automatically with these feedbacks and improves its performance. In reinforcement learning, the agent interacts with the environment and explores it. The goal of an agent is to get the most reward points, and hence, it improves its performance.

The robotic dog, which automatically learns the movement of his arms, is an example of Reinforcement learning.

History of Machine Learning

Before some years (about 40-50 years), machine learning was science fiction, but today it is the part of our daily life. Machine learning is making our day to day life easy from **self-driving cars** to **Amazon virtual assistant "Alexa"**. However, the idea behind machine learning is so old and has a long history. Below some milestones are given which have occurred in the history of machine learning:

The early history of Machine Learning (Pre-1940):

- **1834:** In 1834, Charles Babbage, the father of the computer, conceived a device that could be programmed with punch cards. However, the machine was never built, but all modern computers rely on its logical structure.
- **1936:** In 1936, Alan Turing gave a theory that how a machine can determine and execute a set of instructions.

The era of stored program computers:

- **1940:** In 1940, the first manually operated computer, "ENIAC" was invented, which was the first electronic general-purpose computer. After that stored program computer such as EDSAC in 1949 and EDVAC in 1951 were invented.
- **1943:** In 1943, a human neural network was modeled with an electrical circuit. In 1950, the scientists started applying their idea to work and analyzed how human neurons might work.

Computer machinery and intelligence:

- **1950:** In 1950, Alan Turing published a seminal paper, "**Computer Machinery and Intelligence**," on the topic of artificial intelligence. **In his paper, he asked, "Can machines think?"**

Machine intelligence in Games:

- **1952:** Arthur Samuel, who was the pioneer of machine learning, created a program that helped an IBM computer to play a checkers game. It performed better more it played.
- **1959:** In 1959, the term "Machine Learning" was first coined by **Arthur Samuel**.

Machine Learning from theory to reality

- **1959:** In 1959, the first neural network was applied to a real-world problem to remove echoes over phone lines using an adaptive filter.

- **1985:** In 1985, Terry Sejnowski and Charles Rosenberg invented a neural network **NETtalk**, which was able to teach itself how to correctly pronounce 20,000 words in one week.
- **1997:** The IBM's **Deep blue** intelligent computer won the chess game against the chess expert Garry Kasparov, and it became the first computer which had beaten a human chess expert.

Machine Learning at 21st century

- **2006:** In the year 2006, computer scientist Geoffrey Hinton has given a new name to neural net research as "**deep learning**," and nowadays, it has become one of the most trending technologies.
- **2012:** In 2012, Google created a deep neural network which learned to recognize the image of humans and cats in YouTube videos.
- **2014:** In 2014, the Chabot "**Eugen Goostman**" cleared the Turing Test. It was the first Chabot who convinced the 33% of human judges that it was not a machine.
- **2014: DeepFace** was a deep neural network created by Facebook, and they claimed that it could recognize a person with the same precision as a human can do.
- **2016: AlphaGo** beat the world's number second player **Lee sedol** at **Go game**. In 2017 it beat the number one player of this game **Ke Jie**.
- **2017:** In 2017, the Alphabet's Jigsaw team built an intelligent system that was able to learn the **online trolling**. It used to read millions of comments of different websites to learn to stop online trolling.

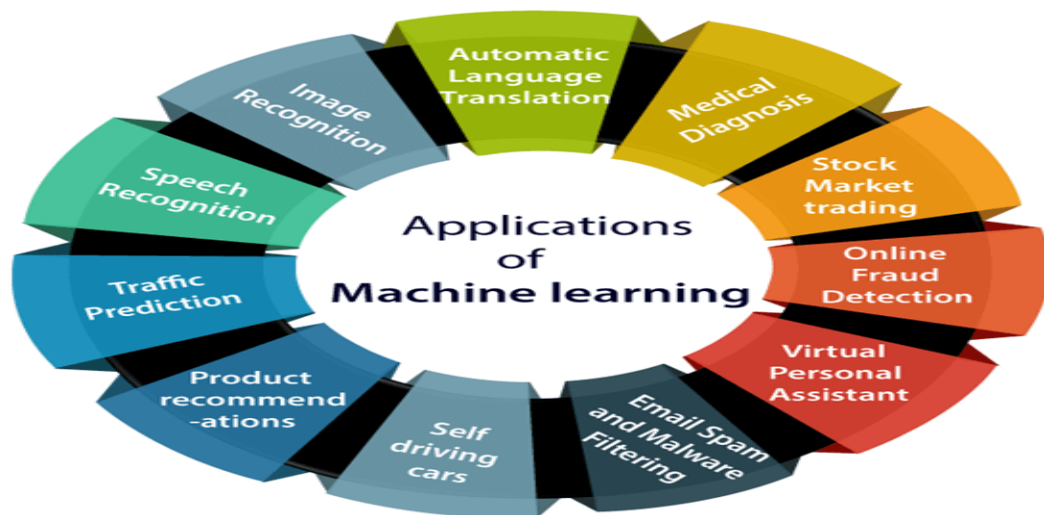
Machine Learning at present:

Now machine learning has got a great advancement in its research, and it is present everywhere around us, such as **self-driving cars, Amazon Alexa, Catboats, recommender system**, and many more. It includes **Supervised, unsupervised, and reinforcement learning with clustering, classification, decision tree, SVM algorithms**, etc.

Modern machine learning models can be used for making various predictions, including **weather prediction, disease prediction, stock market analysis**, etc.

Applications of Machine learning

Machine learning is a buzzword for today's technology, and it is growing very rapidly day by day. We are using machine learning in our daily life even without knowing it such as Google Maps, Google assistant, Alexa, etc. Below are some most trending real-world applications of Machine Learning:



1. Image Recognition:

Image recognition is one of the most common applications of machine learning. It is used to identify objects, persons, places, digital images, etc. The popular use case of image recognition and face detection is, **Automatic friend tagging suggestion**:

Facebook provides us a feature of auto friend tagging suggestion. Whenever we upload a photo with our Facebook friends, then we automatically get a tagging suggestion with name, and the technology behind this is machine learning's **face detection** and **recognition algorithm**.

It is based on the Facebook project named "**Deep Face**," which is responsible for face recognition and person identification in the picture.

2. Speech Recognition

While using Google, we get an option of "**Search by voice**," it comes under speech recognition, and it's a popular application of machine learning.

Speech recognition is a process of converting voice instructions into text, and it is also known as "**Speech to text**", or "**Computer speech recognition**." At present, machine learning algorithms are widely used by various applications of speech recognition. **Google assistant**, **Siri**, **Cortana**, and **Alexa** are using speech recognition technology to follow the voice instructions.

3. Traffic prediction:

If we want to visit a new place, we take help of Google Maps, which shows us the correct path with the shortest route and predicts the traffic conditions.

It predicts the traffic conditions such as whether traffic is cleared, slow-moving, or heavily congested with the help of two ways:

- **Real Time location** of the vehicle from Google Map app and sensors
- **Average time has taken** on past days at the same time.

Everyone who is using Google Map is helping this app to make it better. It takes information from the user and sends back to its database to improve the performance.

4. Product recommendations:

Machine learning is widely used by various e-commerce and entertainment companies such as **Amazon, Netflix**, etc., for product recommendation to the user. Whenever we search for some product on Amazon, then we started getting an advertisement for the same product while internet surfing on the same browser and this is because of machine learning.

Google understands the user interest using various machine learning algorithms and suggests the product as per customer interest.

As similar, when we use Netflix, we find some recommendations for entertainment series, movies, etc., and this is also done with the help of machine learning.

5. Self-driving cars:

One of the most exciting applications of machine learning is self-driving cars. Machine learning plays a significant role in self-driving cars. Tesla, the most popular car manufacturing company is working on self-driving car. It is using unsupervised learning method to train the car models to detect people and objects while driving.

6. Email Spam and Malware Filtering:

Whenever we receive a new email, it is filtered automatically as important, normal, and spam. We always receive an important mail in our inbox with the important symbol and spam emails in our spam box, and the technology behind this is Machine learning. Below are some spam filters used by Gmail:

- Content Filter
- Header filter
- General blacklists filter
- Rules-based filters
- Permission filters

Some machine learning algorithms such as **Multi-Layer Perceptron**, **Decision tree**, and **Naïve Bayes classifier** are used for email spam filtering and malware detection.

7. Virtual Personal Assistant:

We have various virtual personal assistants such as **Google assistant**, **Alexa**, **Cortana**, **Siri**. As the name suggests, they help us in finding the information using our voice instruction. These assistants can help us in various ways just by our voice instructions such as Play music, call someone, Open an email, Scheduling an appointment, etc.

These virtual assistants use machine learning algorithms as an important part.

These assistant record our voice instructions, send it over the server on a cloud, and decode it using ML algorithms and act accordingly.

8. Online Fraud Detection:

Machine learning is making our online transaction safe and secure by detecting fraud transaction. Whenever we perform some online transaction, there may be various ways that a fraudulent transaction can take place such as **fake accounts**, **fake ids**, and **steal money** in the middle of a transaction. So to detect this, **Feed Forward Neural network** helps us by checking whether it is a genuine transaction or a fraud transaction.

For each genuine transaction, the output is converted into some hash values, and these values become the input for the next round. For each genuine transaction, there is a specific pattern which gets change for the fraud transaction hence, it detects it and makes our online transactions more secure.

9. Stock Market trading:

Machine learning is widely used in stock market trading. In the stock market, there is always a risk of up and downs in shares, so for this machine learning's **long short term memory neural network** is used for the prediction of stock market trends.

10. Medical Diagnosis:

In medical science, machine learning is used for diseases diagnoses. With this, medical technology is growing very fast and able to build 3D models that can predict the exact position of lesions in the brain.

It helps in finding brain tumors and other brain-related diseases easily.

11. Automatic Language Translation:

Nowadays, if we visit a new place and we are not aware of the language then it is not a problem at all, as for this also machine learning helps us by converting the text into our known languages. Google's GNMT (Google Neural Machine Translation) provide this feature, which is a Neural Machine Learning that translates the text into our familiar language, and it called as automatic translation.

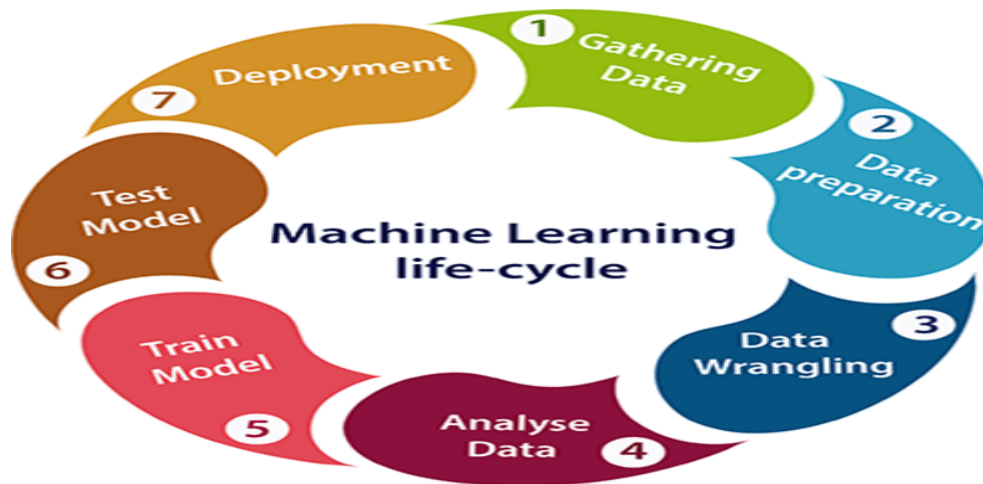
The technology behind the automatic translation is a sequence to sequence learning algorithm, which is used with image recognition and translates the text from one language to another language.

Machine learning Life cycle

Machine learning has given the computer systems the abilities to automatically learn without being explicitly programmed. But how does a machine learning system work? So, it can be described using the life cycle of machine learning. Machine learning life cycle is a cyclic process to build an efficient machine learning project. The main purpose of the life cycle is to find a solution to the problem or project.

Machine learning life cycle involves seven major steps, which are given below:

- **Gathering Data**
- **Data preparation**
- **Data Wrangling**
- **Analyse Data**
- **Train the model**
- **Test the model**
- **Deployment**



The most important thing in the complete process is to understand the problem and to know the purpose of the problem. Therefore, before starting the life cycle, we need to understand the problem because the good result depends on the better understanding of the problem.

In the complete life cycle process, to solve a problem, we create a machine learning system called "model", and this model is created by providing "training". But to train a model, we need data, hence, life cycle starts by collecting data.

1. Gathering Data:

Data Gathering is the first step of the machine learning life cycle. The goal of this step is to identify and obtain all data-related problems.

In this step, we need to identify the different data sources, as data can be collected from various sources such as **files, database, internet, or mobile devices**. It is one of the most important steps of the life cycle. The quantity and quality of the collected data will determine the efficiency of the output. The more will be the data, the more accurate will be the prediction.

This step includes the below tasks:

- **Identify various data sources**
- **Collect data**
- **Integrate the data obtained from different sources**

By performing the above task, we get a coherent set of data, also called as a **dataset**. It will be used in further steps.

2. Data preparation

After collecting the data, we need to prepare it for further steps. Data preparation is a step where we put our data into a suitable place and prepare it to use in our machine learning training.

In this step, first, we put all data together, and then randomize the ordering of data.

This step can be further divided into two processes:

- **Data exploration:**
It is used to understand the nature of data that we have to work with. We need to understand the characteristics, format, and quality of data. A better understanding of data leads to an effective outcome. In this, we find Correlations, general trends, and outliers.
- **Data pre-processing:**
Now the next step is preprocessing of data for its analysis.

3. Data Wrangling

Data wrangling is the process of cleaning and converting raw data into a useable format. It is the process of cleaning the data, selecting the variable to use, and transforming the data in a proper format to make it more suitable for analysis in the next step. It is one of the most important steps of the complete process. Cleaning of data is required to address the quality issues.

It is not necessary that data we have collected is always of our use as some of the data may not be useful. In real-world applications, collected data may have various issues, including:

- **Missing Values**
- **Duplicate data**
- **Invalid data**
- **Noise**

So, we use various filtering techniques to clean the data.

It is mandatory to detect and remove the above issues because it can negatively affect the quality of the outcome.

4. Data Analysis

Now the cleaned and prepared data is passed on to the analysis step. This step involves:

- **Selection of analytical techniques**
- **Building models**
- **Review the result**

The aim of this step is to build a machine learning model to analyze the data using various analytical techniques and review the outcome. It starts with the determination of the type of the problems, where we select the machine learning techniques such as **Classification, Regression, Cluster analysis, Association**, etc. then build the model using prepared data, and evaluate the model.

Hence, in this step, we take the data and use machine learning algorithms to build the model.

5. Train Model

Now the next step is to train the model, in this step we train our model to improve its performance for better outcome of the problem.

We use datasets to train the model using various machine learning algorithms. Training a model is required so that it can understand the various patterns, rules, and, features.

6. Test Model

Once our machine learning model has been trained on a given dataset, then we test the model. In this step, we check for the accuracy of our model by providing a test dataset to it.

Testing the model determines the percentage accuracy of the model as per the requirement of project or problem.

7. Deployment

The last step of machine learning life cycle is deployment, where we deploy the model in the real-world system.

If the above-prepared model is producing an accurate result as per our requirement with acceptable speed, then we deploy the model in the real system. But before deploying the project, we will check whether it is improving its performance using available data or not. The deployment phase is similar to making the final report for a project.

Key differences between Artificial Intelligence (AI) and Machine learning (ML):

Artificial Intelligence	Machine learning
Artificial intelligence is a technology which enables a machine to simulate human behavior.	Machine learning is a subset of AI which allows a machine to automatically learn from past data without programming explicitly.
The goal of AI is to make a smart computer system like humans to solve complex problems.	The goal of ML is to allow machines to learn from data so that they can give accurate output.
In AI, we make intelligent systems to perform any task like a human.	In ML, we teach machines with data to perform a particular task and give an accurate result.
Machine learning and deep learning are the two main subsets of AI.	Deep learning is a main subset of machine learning.
AI has a very wide range of scope.	Machine learning has a limited scope.
AI is working to create an intelligent system which can perform various complex tasks.	Machine learning is working to create machines that can perform only those specific tasks for which they are trained.
AI system is concerned about maximizing the chances of success.	Machine learning is mainly concerned about accuracy and patterns.
The main applications of AI are Siri, customer support using chatbots, Expert System, Online game playing, intelligent humanoid robot, etc.	The main applications of machine learning are Online recommender system, Google search algorithms, Facebook auto friend tagging suggestions, etc.
On the basis of capabilities, AI can be divided into three types, which are, Weak AI, General AI, and Strong AI.	Machine learning can also be divided into mainly three types that are Supervised learning, Unsupervised learning, and Reinforcement learning.
It includes learning, reasoning, and self-correction.	It includes learning and self-correction when introduced with new data.
AI completely deals with Structured, semi-structured, and unstructured data.	Machine learning deals with Structured and semi-structured data.

The main differences between Supervised and Unsupervised learning are given below:

Supervised Learning	Unsupervised Learning
Supervised learning algorithms are trained using labeled data.	Unsupervised learning algorithms are trained using unlabeled data.
Supervised learning model takes direct feedback to check if it is predicting correct output or not.	Unsupervised learning model does not take any feedback.
Supervised learning model predicts the output.	Unsupervised learning model finds the hidden patterns in data.
In supervised learning, input data is provided to the model along with the output.	In unsupervised learning, only input data is provided to the model.
The goal of supervised learning is to train the model so that it can predict the output when it is given new data.	The goal of unsupervised learning is to find the hidden patterns and useful insights from the unknown dataset.
Supervised learning needs supervision to train the model.	Unsupervised learning does not need any supervision to train the model.
Supervised learning can be categorized in Classification and Regression problems.	Unsupervised Learning can be classified in Clustering and Associations problems.
Supervised learning can be used for those cases where we know the input as well as corresponding outputs.	Unsupervised learning can be used for those cases where we have only input data and no corresponding output data.
Supervised learning model produces an accurate result.	Unsupervised learning model may give less accurate result as compared to supervised learning.
Supervised learning is not close to true Artificial intelligence as in this, we first train the model for each data, and then only it can predict the correct output.	Unsupervised learning is more close to the true Artificial Intelligence as it learns similarly as a child learns daily routine things by his experiences.
It includes various algorithms such as Linear Regression, Logistic Regression, Support Vector Machine, Multi-class Classification, Decision tree, Bayesian Logic, etc.	It includes various algorithms such as Clustering, KNN, and Apriori algorithm.

Supervised Machine Learning

Supervised learning is the types of machine learning in which machines are trained using well "labelled" training data, and on basis of that data, machines predict the output. The labelled data means some input data is already tagged with the correct output.

In supervised learning, the training data provided to the machines work as the supervisor that teaches the machines to predict the output correctly. It applies the same concept as a student learns in the supervision of the teacher.

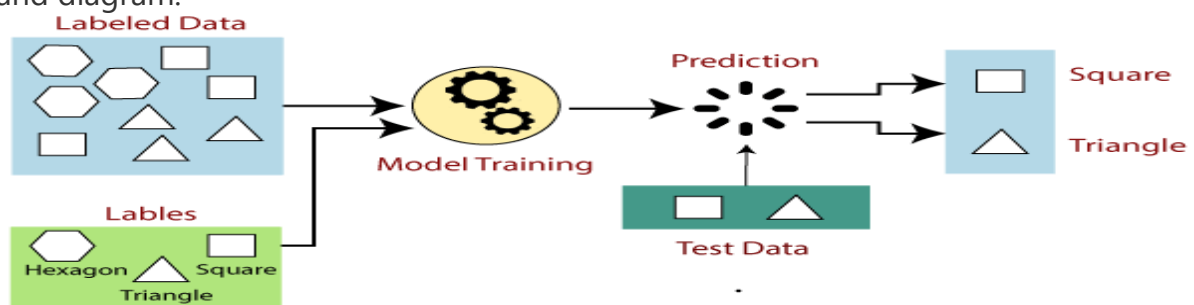
Supervised learning is a process of providing input data as well as correct output data to the machine learning model. The aim of a supervised learning algorithm is to **find a mapping function to map the input variable(x) with the output variable(y).**

In the real-world, supervised learning can be used for **Risk Assessment, Image classification, Fraud Detection, spam filtering, etc.**

How Supervised Learning Works?

In supervised learning, models are trained using labelled dataset, where the model learns about each type of data. Once the training process is completed, the model is tested on the basis of test data (a subset of the training set), and then it predicts the output.

The working of Supervised learning can be easily understood by the below example and diagram:



Suppose we have a dataset of different types of shapes which includes square, rectangle, triangle, and Polygon. Now the first step is that we need to train the model for each shape.

- If the given shape has four sides, and all the sides are equal, then it will be labelled as a **Square**.
- If the given shape has three sides, then it will be labelled as a **triangle**.
- If the given shape has six equal sides then it will be labelled as **hexagon**.

Now, after training, we test our model using the test set, and the task of the model is to identify the shape.

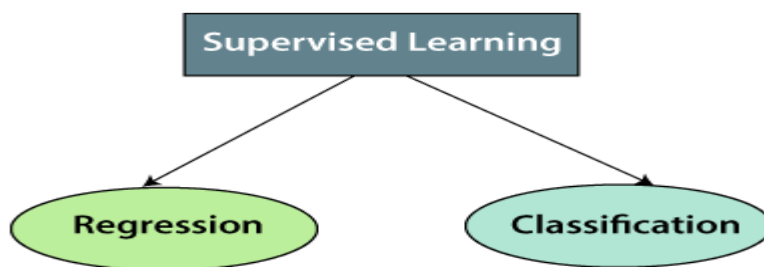
The machine is already trained on all types of shapes, and when it finds a new shape, it classifies the shape on the bases of a number of sides, and predicts the output.

Steps Involved in Supervised Learning:

- First Determine the type of training dataset
- Collect/Gather the labelled training data.
- Split the training dataset into training **dataset**, **test dataset**, and **validation dataset**.
- Determine the input features of the training dataset, which should have enough knowledge so that the model can accurately predict the output.
- Determine the suitable algorithm for the model, such as support vector machine, decision tree, etc.
- Execute the algorithm on the training dataset. Sometimes we need validation sets as the control parameters, which are the subset of training datasets.
- Evaluate the accuracy of the model by providing the test set. If the model predicts the correct output, which means our model is accurate.

Types of supervised Machine learning Algorithms:

Supervised learning can be further divided into two types of problems:



1. Regression

Regression algorithms are used if there is a relationship between the input variable and the output variable. It is used for the prediction of continuous variables, such as Weather forecasting, Market Trends, etc. Below are some popular Regression algorithms which come under supervised learning:

- Linear Regression
- Regression Trees
- Non-Linear Regression
- Bayesian Linear Regression
- Polynomial Regression

2. Classification

Classification algorithms are used when the output variable is categorical, which means there are two classes such as Yes-No, Male-Female, True-false, etc.

Spam Filtering,

- Random Forest
- Decision Trees
- Logistic Regression
- Support vector Machines

Advantages of Supervised learning:

- With the help of supervised learning, the model can predict the output on the basis of prior experiences.
- In supervised learning, we can have an exact idea about the classes of objects.
- Supervised learning model helps us to solve various real-world problems such as **fraud detection, spam filtering**, etc.

Disadvantages of supervised learning:

- Supervised learning models are not suitable for handling the complex tasks.
- Supervised learning cannot predict the correct output if the test data is different from the training dataset.
- Training required lots of computation times.
- In supervised learning, we need enough knowledge about the classes of object.

What is Unsupervised Learning?

As the name suggests, unsupervised learning is a machine learning technique in which models are not supervised using training dataset. Instead, models itself find the hidden patterns and insights from the given data. It can be compared to learning which takes place in the human brain while learning new things. It can be defined as:

Unsupervised learning is a type of machine learning in which models are trained using unlabeled dataset and are allowed to act on that data without any supervision.

Unsupervised learning cannot be directly applied to a regression or classification problem because unlike supervised learning, we have the input data but no

corresponding output data. The goal of unsupervised learning is to **find the underlying structure of dataset, group that data according to similarities, and represent that dataset in a compressed format.**

Example: Suppose the unsupervised learning algorithm is given an input dataset containing images of different types of cats and dogs. The algorithm is never trained upon the given dataset, which means it does not have any idea about the features of the dataset. The task of the unsupervised learning algorithm is to identify the image features on their own. Unsupervised learning algorithm will perform this task by clustering the image dataset into the groups according to similarities between images.

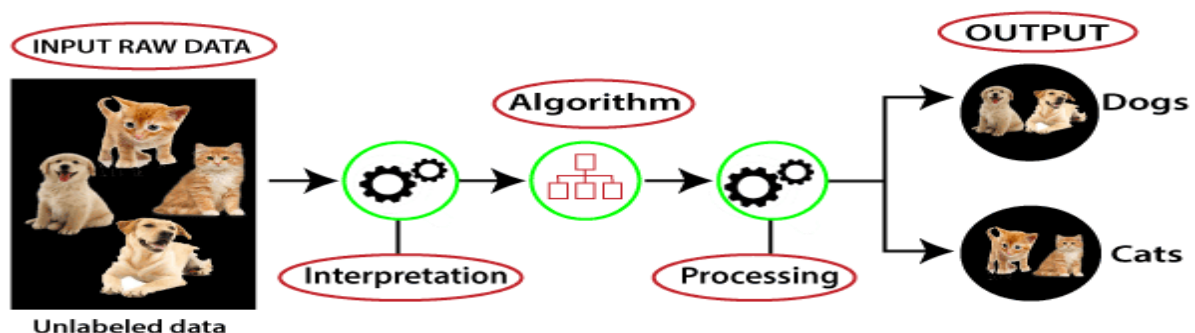
Why use Unsupervised Learning?

Below are some main reasons which describe the importance of Unsupervised Learning:

- Unsupervised learning is helpful for finding useful insights from the data.
- Unsupervised learning is much similar as a human learns to think by their own experiences, which makes it closer to the real AI.
- Unsupervised learning works on unlabeled and uncategorized data which make unsupervised learning more important.
- In real-world, we do not always have input data with the corresponding output so to solve such cases, we need unsupervised learning.

Working of Unsupervised Learning

Working of unsupervised learning can be understood by the below diagram:



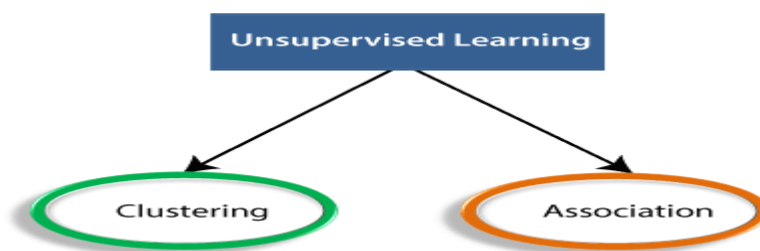
Here, we have taken an unlabeled input data, which means it is not categorized and corresponding outputs are also not given. Now, this unlabeled input data is fed to the machine learning model in order to train it. Firstly, it will interpret the raw data to find

the hidden patterns from the data and then will apply suitable algorithms such as k-means clustering, Decision tree, etc.

Once it applies the suitable algorithm, the algorithm divides the data objects into groups according to the similarities and difference between the objects.

Types of Unsupervised Learning Algorithm:

The unsupervised learning algorithm can be further categorized into two types of problems:



- **Clustering:** Clustering is a method of grouping the objects into clusters such that objects with most similarities remain in a group and have less or no similarities with the objects of another group. Cluster analysis finds the commonalities between the data objects and categorizes them as per the presence and absence of those commonalities.
- **Association:** An association rule is an unsupervised learning method which is used for finding the relationships between variables in the large database. It determines the set of items that occurs together in the dataset. Association rule makes marketing strategy more effective. Such as people who buy X item (suppose a bread) are also tend to purchase Y (Butter/Jam) item. A typical example of Association rule is Market Basket Analysis.

Unsupervised Learning algorithms:

Below is the list of some popular unsupervised learning algorithms:

- K-means clustering
- KNN (k-nearest neighbors)
- Hierarchical clustering
- Anomaly detection
- Neural Networks
- Principle Component Analysis
- Independent Component Analysis
- Apriori algorithm
- Singular value decomposition

Advantages of Unsupervised Learning

- Unsupervised learning is used for more complex tasks as compared to supervised learning because, in unsupervised learning, we don't have labeled input data.
- Unsupervised learning is preferable as it is easy to get unlabeled data in comparison to labeled data.

Disadvantages of Unsupervised Learning

- Unsupervised learning is intrinsically more difficult than supervised learning as it does not have corresponding output.
- The result of the unsupervised learning algorithm might be less accurate as input data is not labeled, and algorithms do not know the exact output in advance.

Classification	Clustering
Classification is a supervised learning approach where a specific label is provided to the machine to classify new observations. Here the machine needs proper testing and training for the label verification.	Clustering is an unsupervised learning approach where grouping is done on similarities basis.
Supervised learning approach.	Unsupervised learning approach.
It uses a training dataset.	It does not use a training dataset.
It uses algorithms to categorize the new data as per the observations of the training set.	It uses statistical concepts in which the data set is divided into subsets with the same features.
In classification, there are labels for training data.	In clustering, there are no labels for training data.
Its objective is to find which class a new object belongs to form the set of predefined classes.	Its objective is to group a set of objects to find whether there is any relationship between them.
It is more complex as compared to clustering.	It is less complex as compared to clustering.

UNIT-02

What is a dataset?

A **dataset** is a collection of data in which data is arranged in some order. A dataset can contain any data from a series of an array to a database table. Below table shows an example of the dataset:

Country	Age	Salary	Purchased
India	38	48000	No
France	43	45000	Yes
Germany	30	54000	No
France	48	65000	No
Germany	40		Yes
India	35	58000	Yes

A tabular dataset can be understood as a database table or matrix, where each column corresponds to a **particular variable**, and each row corresponds to the **fields of the dataset**. The most supported file type for a tabular dataset is "**Comma Separated File**," or **CSV**. But to store a "tree-like data," we can use the JSON file more efficiently.

Types of data in datasets

- **Numerical data:**Such as house price, temperature, etc.
- **Categorical data:**Such as Yes/No, True/False, Blue/green, etc.
- **Ordinal data:**These data are similar to categorical data but can be measured on the basis of comparison.

Note: A real-world dataset is of huge size, which is difficult to manage and process at the initial level. Therefore, to practice machine learning algorithms, we can use any dummy dataset.

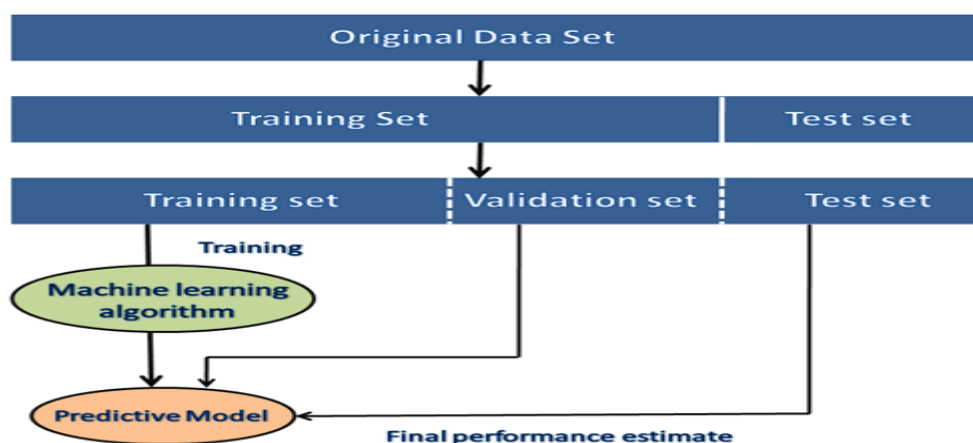
Need of Dataset

To work with machine learning projects, we need a huge amount of data, because, without the data, one cannot train ML/AI models. Collecting and preparing the dataset is one of the most crucial parts while creating an ML/AI project.

The technology applied behind any ML projects cannot work properly if the dataset is not well prepared and pre-processed.

During the development of the ML project, the developers completely rely on the datasets. In building ML applications, datasets are divided into two parts:

- **Training dataset:**
- **Test Dataset**



Note: The datasets are of large size, so to download these datasets, you must have fast internet on your computer.

Popular sources for Machine Learning datasets

Below is the list of datasets which are freely available for the public to work on it:

1. Kaggle Datasets

Kaggle is one of the best sources for providing datasets for Data Scientists and Machine Learners. It allows users to find, download, and publish datasets in an easy way. It also provides the opportunity to work with other machine learning engineers and solve difficult Data Science related tasks.

Kaggle provides a high-quality dataset in different formats that we can easily find and download.

The link for the Kaggle dataset is <https://www.kaggle.com/datasets>.

2. UCI Machine Learning Repository

UCI Machine learning repository is one of the great sources of machine learning datasets. This repository contains databases, domain theories, and data generators that are widely used by the machine learning community for the analysis of ML algorithms.

Since the year 1987, it has been widely used by students, professors, researchers as a primary source of machine learning dataset.

It classifies the datasets as per the problems and tasks of machine learning such as **Regression, Classification, Clustering, etc.** It also contains some of the popular datasets such as the **Iris dataset, Car Evaluation dataset, Poker Hand dataset, etc.**

The link for the UCI machine learning repository is <https://archive.ics.uci.edu/ml/index.php>.

3. Datasets via AWS

We can search, download, access, and share the datasets that are publicly available via AWS resources. These datasets can be accessed through AWS resources but provided and maintained by different government organizations, researches, businesses, or individuals.

Anyone can analyze and build various services using shared data via AWS resources. The shared dataset on cloud helps users to spend more time on data analysis rather than on acquisitions of data.

This source provides the various types of datasets with examples and ways to use the dataset. It also provides the search box using which we can search for the required dataset. Anyone can add any dataset or example to the **Registry of Open Data on AWS**.

The link for the resource is <https://registry.opendata.aws/>.

4. Google's Dataset Search Engine

Google dataset search engine is a search engine launched by **Google** on **September 5, 2018**. This source helps researchers to get online datasets that are freely available for use.

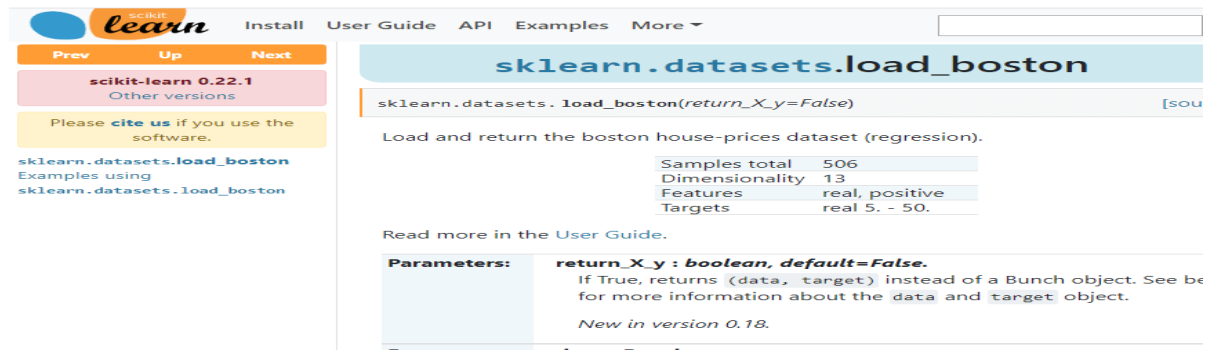
The link for the Google dataset search engine is <https://toolbox.google.com/datasetsearch>.

7. Government Datasets

There are different sources to get government-related data. Various countries publish government data for public use collected by them from different departments.

- [Indian Government dataset](#)
- [US Government Dataset](#)
- [Northern Ireland Public Sector Datasets](#)
- [European Union Open Data Portal](#)

9. Scikit-learn dataset



The screenshot shows the scikit-learn website's documentation for the `sklearn.datasets.load_boston` function. The page includes a navigation bar with links to 'Install', 'User Guide', 'API', 'Examples', and 'More'. A sidebar on the left contains links to 'Prev', 'Up', and 'Next', along with version information for 'scikit-learn 0.22.1' and a 'cite us' notice. The main content area displays the function signature `sklearn.datasets.load_boston(return_X_y=False)` and a description: 'Load and return the boston house-prices dataset (regression)'. A table lists dataset statistics: Samples total (506), Dimensionality (13), Features (real, positive), and Targets (real 5. - 50.). Below this, the 'Parameters' section shows `return_X_y : boolean, default=False` with a detailed explanation. The 'Returns' section indicates `data : Bunch`. A link to 'Read more in the User Guide' is also present.

Statistics	Value
Samples total	506
Dimensionality	13
Features	real, positive
Targets	real 5. - 50.

Scikit-learn is a great source for machine learning enthusiasts. This source provides both toy and real-world datasets. These datasets can be obtained from `sklearn.datasets` package and using general dataset API.

The toy dataset available on scikit-learn can be loaded using some predefined functions such as, `load_boston([return_X_y])`, `load_iris([return_X_y])`, etc, rather than importing any file from external sources. But these datasets are not suitable for real-world projects.

Data Preprocessing in Machine learning

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model.

When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So for this, we use data preprocessing task.

Why do we need Data Preprocessing?

A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data preprocessing is required tasks for cleaning the data and making it suitable for a

machine learning model which also increases the accuracy and efficiency of a machine learning model.

It involves below steps:

- **Getting the dataset**
- **Importing libraries**
- **Importing datasets**
- **Finding Missing Data**
- **Encoding Categorical Data**
- **Splitting dataset into training and test set**
- **Feature scaling**

1) Get the Dataset

To create a machine learning model, the first thing we required is a dataset as a machine learning model completely works on data. The collected data for a particular problem in a proper format is known as the **dataset**.

Dataset may be of different formats for different purposes, such as, if we want to create a machine learning model for business purpose, then dataset will be different with the dataset required for a liver patient. So each dataset is different from another dataset. To use the dataset in our code, we usually put it into a CSV **file**. However, sometimes, we may also need to use an HTML or xlsx file.

What is a CSV File?

CSV stands for "**Comma-Separated Values**" files; it is a file format which allows us to save the tabular data, such as spreadsheets. It is useful for huge datasets and can use these datasets in programs.

Here we will use a demo dataset for data preprocessing, and for practice, it can be downloaded from here, "<https://www.superdatascience.com/pages/machine-learning>". For real-world problems, we can download datasets online from various sources such as <https://www.kaggle.com/uciml/datasets>, <https://archive.ics.uci.edu/ml/index.php> etc.

We can also create our dataset by gathering data using various API with Python and put that data into a .csv file.

2) Importing Libraries

In order to perform data preprocessing using Python, we need to import some predefined Python libraries. These libraries are used to perform some specific jobs. There are three specific libraries that we will use for data preprocessing, which are:

Numpy: Numpy Python library is used for including any type of mathematical operation in the code. It is the fundamental package for scientific calculation in Python. It also supports to add large, multidimensional arrays and matrices. So, in Python, we can import it as:

1. `import numpy as nm`

Here we have used **nm**, which is a short name for Numpy, and it will be used in the whole program.

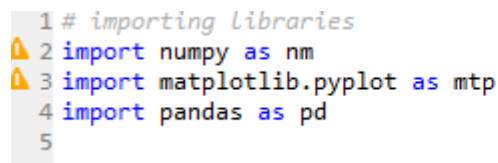
Matplotlib: The second library is **matplotlib**, which is a Python 2D plotting library, and with this library, we need to import a sub-library **pyplot**. This library is used to plot any type of charts in Python for the code. It will be imported as below:

1. `import matplotlib.pyplot as mtp`

Here we have used **mtp** as a short name for this library.

Pandas: The last library is the Pandas library, which is one of the most famous Python libraries and used for importing and managing the datasets. It is an open-source data manipulation and analysis library. It will be imported as below:

Here, we have used **pd** as a short name for this library. Consider the below image:



```
1 # importing libraries
2 import numpy as nm
3 import matplotlib.pyplot as mtp
4 import pandas as pd
5
```

3) Importing the Datasets

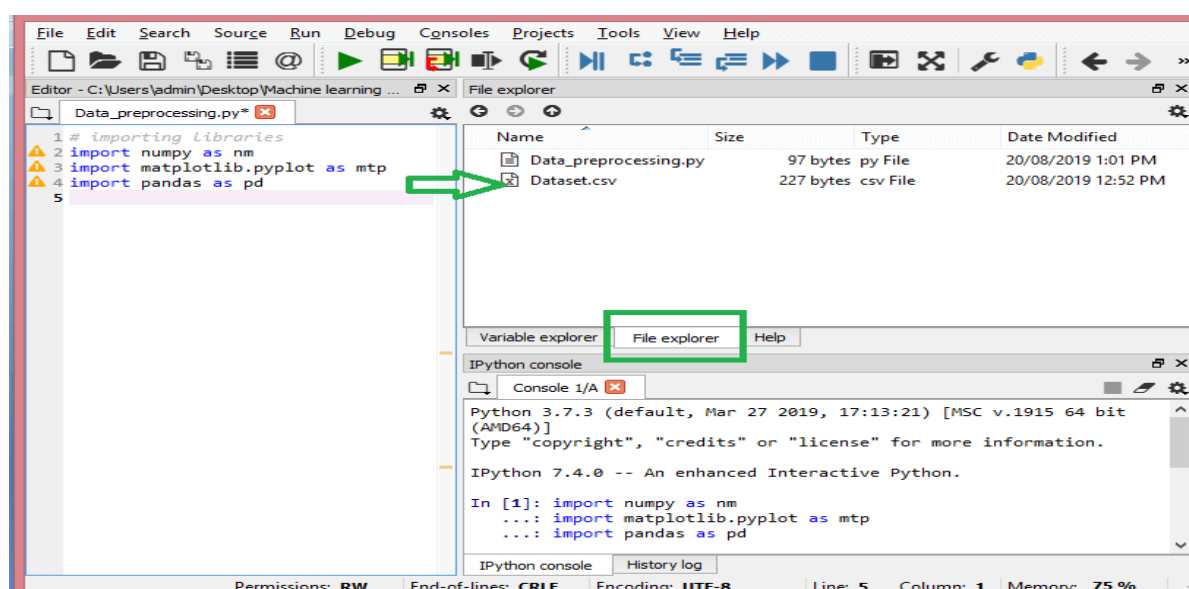
Now we need to import the datasets which we have collected for our machine learning project. But before importing a dataset, we need to set the current directory as a working directory. To set a working directory in Spyder IDE, we need to follow the below steps:

1. Save your Python file in the directory which contains dataset.
2. Go to File explorer option in Spyder IDE, and select the required directory.

3. Click on F5 button or run option to execute the file.

Note: We can set any directory as a working directory, but it must contain the required dataset.

Here, in the below image, we can see the Python file along with required dataset. Now, the current folder is set as a working directory.



read_csv() function:

Now to import the dataset, we will use `read_csv()` function of pandas library, which is used to read a csv file and performs various operations on it. Using this function, we can read a csv file locally as well as through an URL.

We can use `read_csv` function as below:

1. `data_set= pd.read_csv('Dataset.csv')`

Here, **data_set** is a name of the variable to store our dataset, and inside the function, we have passed the name of our dataset. Once we execute the above line of code, it will successfully import the dataset in our code. We can also check the imported dataset by clicking on the section **variable explorer**, and then double click on **data_set**. Consider the below image:

Extracting dependent and independent variables:

In machine learning, it is important to distinguish the matrix of features (independent variables) and dependent variables from dataset. In our dataset, there are three independent variables that are **Country**, **Age**, and **Salary**, and one is a dependent variable which is **Purchased**.

Extracting independent variable:

To extract an independent variable, we will use **iloc[]** method of Pandas library. It is used to extract the required rows and columns from the dataset.

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

In the above code, the first colon(:) is used to take all the rows, and the second colon(:) is for all the columns. Here we have used :-1, because we don't want to take the last column as it contains the dependent variable. So by doing this, we will get the matrix of features.

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

```
1. [['India' 38.0 68000.0]
2. ['France' 43.0 45000.0]
3. ['Germany' 30.0 54000.0]
4. ['France' 48.0 65000.0]
5. ['Germany' 40.0 nan]
6. ['India' 35.0 58000.0]
7. ['Germany' nan 53000.0]
8. ['France' 49.0 79000.0]
9. ['India' 50.0 88000.0]
10. ['France' 37.0 77000.0]]
```

As we can see in the above output, there are only three variables.

Extracting dependent variable:

To extract dependent variables, again, we will use Pandas .iloc[] method.

```
1. y= data_set.iloc[:,3].values
```

Here we have taken all the rows with the last column only. It will give the array of dependent variables.

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

Output:

```
array(['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes'],
      dtype=object)
```

Note: If you are using Python language for machine learning, then extraction is mandatory, but for R language it is not required.

4) Handling Missing data:

The next step of data preprocessing is to handle missing data in the datasets. If our dataset contains some missing data, then it may create a huge problem for our machine learning model. Hence it is necessary to handle missing values present in the dataset.

Ways to handle missing data:

There are mainly two ways to handle missing data, which are:

By deleting the particular row: The first way is used to commonly deal with null values. In this way, we just delete the specific row or column which consists of null values. But this way is not so efficient and removing data may lead to loss of information which will not give the accurate output.

By calculating the mean: In this way, we will calculate the mean of that column or row which contains any missing value and will put it on the place of missing value. This strategy is useful for the features which have numeric data such as age, salary, year, etc. Here, we will use this approach.

To handle missing values, we will use **Scikit-learn** library in our code, which contains various libraries for building machine learning models. Here we will use **Imputer** class of **sklearn.preprocessing** library. Below is the code for it:

1. #handling missing data (Replacing missing data with the mean value)
2. from sklearn.preprocessing import Imputer
3. `imputer= Imputer(missing_values='NaN', strategy='mean', axis = 0)`
4. #Fitting imputer object to the independent variables x.
5. `imputerimputer= imputer.fit(x[:, 1:3])`
6. #Replacing missing data with the calculated mean value
7. `x[:, 1:3]= imputer.transform(x[:, 1:3])`

Output:

```
array([[ 'India', 38.0, 68000.0],
       [ 'France', 43.0, 45000.0],
       [ 'Germany', 30.0, 54000.0],
       [ 'France', 48.0, 65000.0],
       [ 'Germany', 40.0, 65222.22222222222],
       [ 'India', 35.0, 58000.0],
       [ 'Germany', 41.111111111111114, 53000.0],
       [ 'France', 49.0, 79000.0],
       [ 'India', 50.0, 88000.0],
       [ 'France', 37.0, 77000.0]], dtype=object)
```

As we can see in the above output, the missing values have been replaced with the means of rest column values.

5) Encoding Categorical data:

Categorical data is data which has some categories such as, in our dataset; there are two categorical variable, **Country**, and **Purchased**.

Since machine learning model completely works on mathematics and numbers, but if our dataset would have a categorical variable, then it may create trouble while building the model. So it is necessary to encode these categorical variables into numbers.

For Country variable:

Firstly, we will convert the country variables into categorical data. So to do this, we will use **LabelEncoder()** class from **preprocessing** library.

1. #Categorical data
2. #for Country Variable
3. from sklearn.preprocessing import LabelEncoder
4. `label_encoder_x= LabelEncoder()`
5. `x[:, 0]= label_encoder_x.fit_transform(x[:, 0])`

Output:

```
Out[15]:
array([[2, 38.0, 68000.0],
       [0, 43.0, 45000.0],
       [1, 30.0, 54000.0],
       [0, 48.0, 65000.0],
       [1, 40.0, 65222.22222222222],
       [2, 35.0, 58000.0],
       [1, 41.111111111111114, 53000.0],
       [0, 49.0, 79000.0],
       [2, 50.0, 88000.0],
       [0, 37.0, 77000.0]], dtype=object)
```

Explanation:

In above code, we have imported **LabelEncoder** class of **sklearn library**. This class has successfully encoded the variables into digits.

But in our case, there are three country variables, and as we can see in the above output, these variables are encoded into 0, 1, and 2. By these values, the machine learning model may assume that there is some correlation between these variables which will produce the wrong output. So to remove this issue, we will use **dummy encoding**.

Dummy Variables:

Dummy variables are those variables which have values 0 or 1. The 1 value gives the presence of that variable in a particular column, and rest variables become 0. With

dummy encoding, we will have a number of columns equal to the number of categories.

In our dataset, we have 3 categories so it will produce three columns having 0 and 1 values. For Dummy Encoding, we will use **OneHotEncoder** class of **preprocessing** library.

1. #for Country Variable
2. from sklearn.preprocessing import LabelEncoder, OneHotEncoder
3. `label_encoder_x= LabelEncoder()`
4. `x[:, 0]= label_encoder_x.fit_transform(x[:, 0])`
5. #Encoding for dummy variables
6. `onehot_encoder= OneHotEncoder(categorical_features= [0])`
7. `x= onehot_encoder.fit_transform(x).toarray()`

Output:

```
array([[0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 3.80000000e+01,
        6.80000000e+04],
       [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 4.30000000e+01,
        4.50000000e+04],
       [0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 3.00000000e+01,
        5.40000000e+04],
       [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 4.80000000e+01,
        6.50000000e+04],
       [0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 4.00000000e+01,
        6.52222222e+04],
       [0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 3.50000000e+01,
        5.80000000e+04],
       [0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 4.11111111e+01,
        5.30000000e+04],
       [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 4.90000000e+01,
        7.90000000e+04],
       [0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 5.00000000e+01,
        8.80000000e+04],
       [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 3.70000000e+01,
        7.70000000e+04]])
```

As we can see in the above output, all the variables are encoded into numbers 0 and 1 and divided into three columns.

It can be seen more clearly in the variables explorer section, by clicking on x option as:

	0	1	2	3	4
0	0	0	1	38	68000
1	1	0	0	43	45000
2	0	1	0	30	54000
3	1	0	0	48	65000
4	0	1	0	40	65222.2
5	0	0	1	35	58000
6	0	1	0	41.1111	53000
7	1	0	0	49	79000
8	0	0	1	50	88000
9	1	0	0	37	77000

For Purchased Variable:

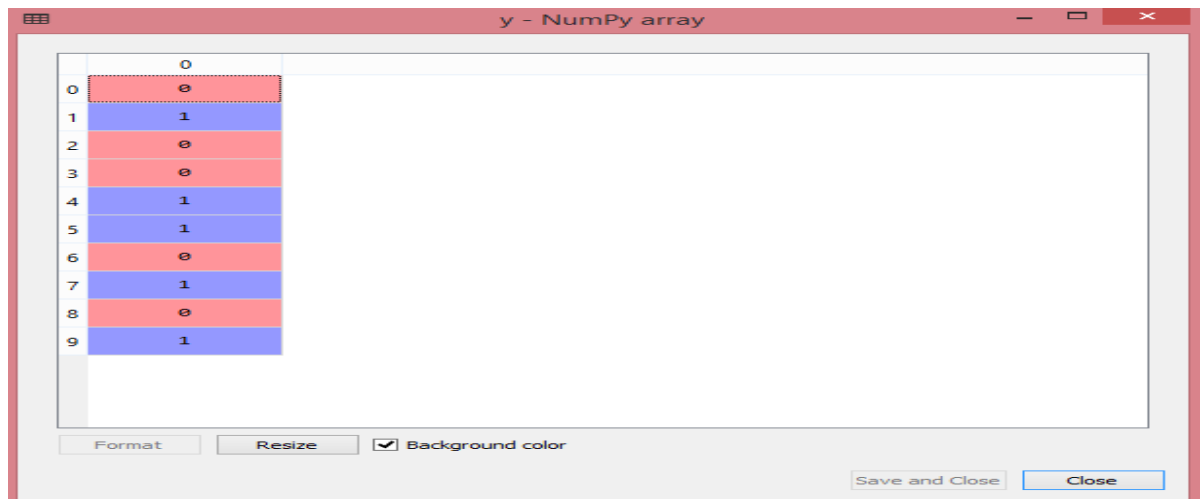
1. `labelencoder_y = LabelEncoder()`
2. `y = labelencoder_y.fit_transform(y)`

For the second categorical variable, we will only use labelencoder object of **LabelEncoder** class. Here we are not using **OneHotEncoder** class because the purchased variable has only two categories yes or no, and which are automatically encoded into 0 and 1.

Output:

```
Out[17]: array([0, 1, 0, 0, 1, 1, 0, 1, 0, 1])
```

It can also be seen as:



6) Splitting the Dataset into the Training set and Test set

In machine learning data preprocessing, we divide our dataset into a training set and test set. This is one of the crucial steps of data preprocessing as by doing this, we can enhance the performance of our machine learning model.

Suppose, if we have given training to our machine learning model by a dataset and we test it by a completely different dataset. Then, it will create difficulties for our model to understand the correlations between the models.

If we train our model very well and its training accuracy is also very high, but we provide a new dataset to it, then it will decrease the performance. So we always try to make a machine learning model which performs well with the training set and also with the test dataset. Here, we can define these datasets as:



Training Set: A subset of dataset to train the machine learning model, and we already know the output.

Test set: A subset of dataset to test the machine learning model, and by using the test set, model predicts the output.

For splitting the dataset, we will use the below lines of code:

1. `from sklearn.model_selection import train_test_split`
2. `x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.2, random_state=0)`

Explanation:

- In the above code, the first line is used for splitting arrays of the dataset into random train and test subsets.
- In the second line, we have used four variables for our output that are
 - **x_train:** features for the training data
 - **x_test:** features for testing data
 - **y_train:** Dependent variables for training data
 - **y_test:** Independent variable for testing data
- In **train_test_split() function**, we have passed four parameters in which first two are for arrays of data, and **test_size** is for specifying the size of the test set. The **test_size** maybe .5, .3, or .2, which tells the dividing ratio of training and testing sets.
- The last parameter **random_state** is used to set a seed for a random generator so that you always get the same result, and the most used value for this is 42.

Output:

By executing the above code, we will get 4 different variables, which can be seen under the variable explorer section.

Variable explorer			
Name	Type	Size	Value
data_set	DataFrame	(10, 4)	Column names: Country, Age, Salary, Purchased
x	float64	(10, 5)	[[0.0e+00 0.0e+00 1.0e+00 3.8e+01 6.8e+04] [1.0e+00 0.0e+00 0.0e+00 4 ...
x_test	float64	(2, 5)	[[0.0e+00 1.0e+00 0.0e+00 3.0e+01 5.4e+04] [0.0e+00 0.0e+00 1.0e+00 5 ...
x_train	float64	(8, 5)	[[0.00000000e+00 1.00000000e+00 0.00000000e+00 4.00000000e+01 6.5222 ...
y	int32	(10,)	[0 1 0 0 1 1 0 1 0 1]
y_test	int32	(2,)	[0 0]
y_train	int32	(8,)	[1 1 1 0 1 0 0 1]

As we can see in the above image, the x and y variables are divided into 4 different variables with corresponding values.

7) Feature Scaling

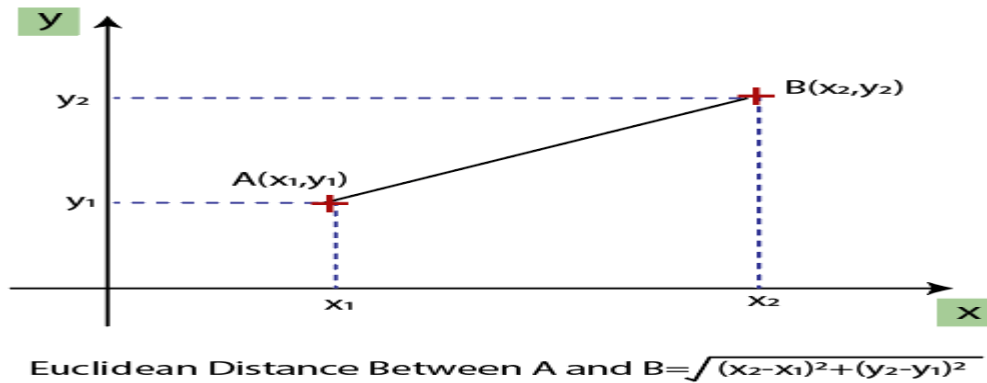
Feature scaling is the final step of data preprocessing in machine learning. It is a technique to standardize the independent variables of the dataset in a specific range. In feature scaling, we put our variables in the same range and in the same scale so that no any variable dominate the other variable.

Consider the below dataset:

data_set - DataFrame					
Index	Country	Age	Salary	Purchased	
0	India	38	68000	No	
1	France	43	45000	Yes	
2	Germany	30	54000	No	
3	France	48	65000	No	
4	Germany	40	nan	Yes	
5	India	35	58000	Yes	
6	Germany	nan	53000	No	
7	France	49	79000	Yes	
8	India	50	88000	No	
9	France	37	77000	Yes	

As we can see, the age and salary column values are not on the same scale. A machine learning model is based on **Euclidean distance**, and if we do not scale the variable, then it will cause some issue in our machine learning model.

Euclidean distance is given as:



If we compute any two values from age and salary, then salary values will dominate the age values, and it will produce an incorrect result. So to remove this issue, we need to perform feature scaling for machine learning.

There are two ways to perform feature scaling in machine learning:

Standardization

$$\text{new value} \rightarrow x' = \frac{x - \text{mean}(x)}{a} \leftarrow \begin{array}{l} \text{original value} \\ \text{mean} \\ \text{Standard deviation} \end{array}$$

Normalization

$$\text{new value} \rightarrow x' = \frac{x - \min(x)}{\max(x) - \min(x)} \leftarrow \text{original value}$$

Here, we will use the standardization method for our dataset.

For feature scaling, we will import **StandardScaler** class of **sklearn.preprocessing** library as:

1. `from sklearn.preprocessing import StandardScaler`

Now, we will create the object of **StandardScaler** class for independent variables or features. And then we will fit and transform the training dataset.

1. `st_x = StandardScaler()`
2. `x_train = st_x.fit_transform(x_train)`

For test dataset, we will directly apply **transform()** function instead of **fit_transform()** because it is already done in training set.

1. `x_test = st_x.transform(x_test)`

Output:

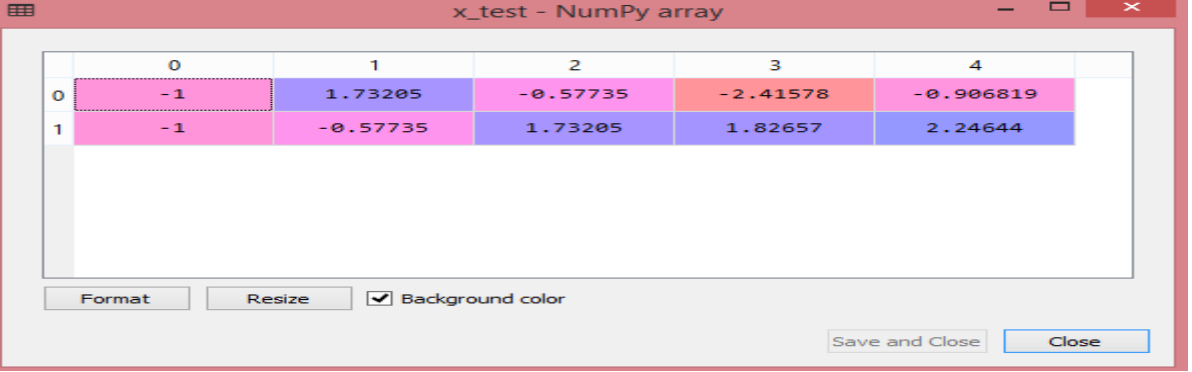
By executing the above lines of code, we will get the scaled values for `x_train` and `x_test` as:

`x_train`:



	0	1	2	3	4
0	-1	1.73205	-0.57735	-0.294607	0.133962
1	1	-0.57735	-0.57735	-0.930959	1.22627
2	1	-0.57735	-0.57735	0.341745	-1.7415
3	-1	1.73205	-0.57735	-0.0589215	-0.999562
4	1	-0.57735	-0.57735	1.61445	1.41175
5	1	-0.57735	-0.57735	1.40233	0.113352
6	-1	-0.57735	1.73205	-0.718842	0.391581
7	-1	-0.57735	1.73205	-1.35519	-0.535848

`x_test`:



	0	1	2	3	4
0	-1	1.73205	-0.57735	-2.41578	-0.906819
1	-1	-0.57735	1.73205	1.82657	2.24644

As we can see in the above output, all the variables are scaled between values -1 to 1.

Note: Here, we have not scaled the dependent variable because there are only two values 0 and 1. But if these variables will have more range of values, then we will also need to scale those variables.

Feature scaling is one of the most important data preprocessing step in machine learning. Algorithms that compute the distance between the features are biased towards numerically larger values if the data is not scaled.

Tree-based algorithms are fairly insensitive to the scale of the features. Also, feature scaling helps machine learning, and deep learning algorithms train and converge faster.

There are some feature scaling techniques such as Normalization and Standardization that are the most popular and at the same time, the most confusing ones.

Let's resolve that confusion.

Normalization or Min-Max Scaling is used to transform features to be on a similar scale. The new point is calculated as:

$$X_{\text{new}} = (X - X_{\text{min}}) / (X_{\text{max}} - X_{\text{min}})$$

This scales the range to [0, 1] or sometimes [-1, 1]. Geometrically speaking, transformation squishes the n-dimensional data into an n-dimensional unit hypercube. Normalization is useful when there are no outliers as it cannot cope up with them. Usually, we would scale age and not incomes because only a few people have high incomes but the age is close to uniform.

Standardization or Z-Score Normalization is the transformation of features by subtracting from mean and dividing by standard deviation. This is often called as Z-score.

$$X_{\text{new}} = (X - \text{mean}) / \text{Std}$$

Standardization can be helpful in cases where the data follows a Gaussian distribution. However, this does not have to be necessarily true. Geometrically speaking, it translates the data to the mean vector of original data to the origin and squishes or expands the points if std is 1 respectively. We can see that we are just changing mean and standard deviation to a standard normal distribution which is still normal thus the shape of the distribution is not affected.

Standardization does not get affected by outliers because there is no predefined range of transformed features.

Difference between Normalization and Standardization

S.NO.	Normalization	Standardization
1.	Minimum and maximum value of features are used for scaling	Mean and standard deviation is used for scaling.
2.	It is used when features are of different scales.	It is used when we want to ensure zero mean and unit standard deviation.
3.	Scales values between [0, 1] or [-1, 1].	It is not bounded to a certain range.
4.	It is really affected by outliers.	It is much less affected by outliers.
5.	Scikit-Learn provides a transformer called <code>MinMaxScaler</code> for Normalization.	Scikit-Learn provides a transformer called <code>StandardScaler</code> for standardization.
6.	This transformation squishes the n-dimensional data into an n-dimensional unit hypercube.	It translates the data to the mean vector of original data to the origin and squishes or expands.

S.NO.	Normalization	Standardization
7.	It is useful when we don't know about the distribution	It is useful when the feature distribution is Normal or Gaussian.
8.	It is a often called as Scaling Normalization	It is a often called as Z-Score Normalization.

What is Feature Scaling?

Feature scaling is a data preprocessing technique that involves transforming the values of features or variables in a dataset to a similar scale. This is done to ensure that all features contribute equally to the model and to prevent features with larger values from dominating the model. Feature scaling is essential when working with datasets where the features have different ranges, units of measurement, or orders of magnitude. Common feature scaling techniques include standardization, normalization, and min-max scaling. By applying feature scaling, the data can be transformed to a more consistent scale, making it easier to build accurate and effective machine learning models.

Why Should We Use Feature Scaling?

Some machine learning algorithms are sensitive to feature scaling, while others are virtually invariant. Let's explore these in more depth:

1. Gradient Descent Based Algorithms

Machine learning algorithms like **linear regression**, **logistic regression**, **neural network**, PCA (principal component analysis), etc., that use gradient descent as an optimization technique require data to be scaled. Take a look at the formula for gradient descent below:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

The presence of feature value X in the formula will affect the step size of the gradient descent. The difference in the ranges of features will cause different step sizes for each feature. To ensure that the gradient descent moves smoothly towards the minima and that the steps for gradient descent are updated at the same rate for all the features, we scale the data before feeding it to the model.

Having features on a similar scale can help the gradient descent converge more quickly towards the minima.

2. Distance-Based Algorithms

Distance algorithms like [KNN](#), [K-means clustering](#), and [SVM](#)(support vector machines) are most affected by the range of features. This is because, behind the scenes, **they are using distances between data points to determine their similarity.**

For example, let's say we have data containing high school CGPA scores of students (ranging from 0 to 5) and their future incomes (in thousands Rupees):

	Student	CGPA	Salary '000
0	1	3.0	60
1	2	3.0	40
2	3	4.0	40
3	4	4.5	50
4	5	4.2	52

Since both the features have different scales, there is a chance that higher weightage is given to features with higher magnitudes. This will impact the performance of the machine learning algorithm; obviously, we do not want our algorithm to be biased towards one feature.

Therefore, we scale our data before employing a distance based algorithm so that all the features contribute equally to the result.

	Student	CGPA	Salary '000
0	1	-1.184341	1.520013
1	2	-1.184341	-1.100699
2	3	0.416120	-1.100699
3	4	1.216350	0.209657
4	5	0.736212	0.471728

The effect of scaling is conspicuous when we compare the Euclidean distance between data points for students A and B, and between B and C, before and after scaling, as shown below:

- Distance AB before scaling $\Rightarrow \sqrt{(40 - 60)^2 + (3 - 3)^2} = 20$
- Distance BC before scaling $\Rightarrow \sqrt{(40 - 40)^2 + (4 - 3)^2} = 1$
- Distance AB after scaling $\Rightarrow \sqrt{(1.1 + 1.5)^2 + (1.18 - 1.18)^2} = 2.6$
- Distance BC after scaling $\Rightarrow \sqrt{(1.1 - 1.1)^2 + (0.41 + 1.18)^2} = 1.59$

3. Tree-Based Algorithms

[Tree-based algorithms](#), on the other hand, are fairly insensitive to the scale of the features. Think about it, a decision tree only splits a node based on a single feature. The decision tree splits a node on a feature that increases the homogeneity of the node. Other features do not influence this split on a feature.

So, the remaining features have virtually no effect on the split. This is what makes them invariant to the scale of the features!

What Is Normalization?

Normalization is a data preprocessing technique used to adjust the values of features in a dataset to a common scale. This is done to facilitate data analysis and modeling, and to reduce the impact of different scales on the accuracy of machine learning models.

Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling.

Here's the formula for normalization:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Here, Xmax and Xmin are the maximum and the minimum values of the feature, respectively.

- When the value of X is the minimum value in the column, the numerator will be 0, and hence X' is 0
- On the other hand, when the value of X is the maximum value in the column, the numerator is equal to the denominator, and thus the value of X' is 1
- If the value of X is between the minimum and the maximum value, then the value of X' is between 0 and 1

What Is Standardization?

Standardization is another scaling method where the values are centered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero, and the resultant distribution has a unit standard deviation.

Here's the formula for standardization:

$$X' = \frac{X - \mu}{\sigma}$$

μ is the mean of the feature values and σ is the standard deviation of the feature values. Note that, in this case, the values are not restricted to a particular range.

Now, the big question in your mind must be when should we use normalization and when should we use standardization? Let's find out!

The Big Question – Normalize or Standardize?

Normalization	Standardization
Rescales values to a range between 0 and 1	Centers data around the mean and scales to a standard deviation of 1
Useful when the distribution of the data is unknown or not Gaussian	Useful when the distribution of the data is Gaussian or unknown
Sensitive to outliers	Less sensitive to outliers
Retains the shape of the original distribution	Changes the shape of the original distribution
May not preserve the relationships between the data points	Preserves the relationships between the data points
Equation: $(x - \min) / (\max - \min)$	Equation: $(x - \text{mean}) / \text{standard deviation}$

However, at the end of the day, the choice of using normalization or standardization will depend on your problem and the machine learning algorithm you are using. There is no hard and fast rule to tell you when to normalize or standardize your data. **You can always start by fitting your model to raw, normalized, and standardized data and comparing the performance for the best results.** *It is a good practice to fit the scaler on the training data and then use it to transform the testing data. This would avoid any data leakage during the model testing process. Also, the scaling of target values is generally not required.*

Implementing Feature Scaling in Python

Python Code:

`pd.describe()` method:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Tier 2	Tier 3	Supermarket Type1	Supermarket Type2
count	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000
mean	12.835420	0.355675	-2.940445	140.486413	15.154884	0.830302	0.326049	0.396571	0.651071	0.111616
std	4.233450	0.478753	0.791551	52.067053	8.383349	0.596352	0.468800	0.489009	0.476567	0.314917
min	4.555000	0.000000	-5.633875	31.290000	4.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	9.300000	0.000000	-3.467344	93.385700	9.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	12.857545	0.000000	-2.862535	142.179900	14.000000	1.000000	0.000000	0.000000	1.000000	0.000000
75%	16.000000	1.000000	-2.331264	184.495000	26.000000	1.000000	1.000000	1.000000	1.000000	0.000000
max	21.350000	1.000000	-1.113550	265.880400	28.000000	2.000000	1.000000	1.000000	1.000000	1.000000

We can see that there is a huge difference in the range of values present in our numerical features: **Item_Visibility**, **Item_Weight**, **Item_MRP**, and **Outlet_Establishment_Year**. Let's try and fix that using feature scaling!

Normalization Using sklearn (scikit-learn)

To normalize your data, you need to import the `MinMaxScaler` from the [sklearn](https://scikit-learn.org/stable/) library and apply it to our dataset. So, let's do that!

```
# data normalization with sklearn

from sklearn.preprocessing import MinMaxScaler

# fit scaler on training data

norm = MinMaxScaler().fit(X_train)

# transform training data

X_train_norm = norm.transform(X_train)

# transform testing dataabs

X_test_norm = norm.transform(X_test)
```

Let's see how normalization has affected our dataset:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Tier 2	Tier 3	Supermarket type1	Supermarket type2
count	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000
mean	0.493029	0.355676	0.595849	0.463485	0.464787	0.416151	0.326049	0.395571	0.651071	0.111616
std	0.252055	0.478753	0.175109	0.263444	0.349556	0.299176	0.458800	0.489009	0.475667	0.314917
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.282525	0.000000	0.479154	0.263556	0.208333	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.494352	0.000000	0.613084	0.470573	0.416667	0.500000	0.000000	0.000000	1.000000	0.000000
75%	0.681453	1.000000	0.730614	0.650250	0.916667	0.500000	1.000000	1.000000	1.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

All the features now have a minimum value of 0 and a maximum value of 1. Perfect!

Next, let's try to standardize our data.

Standardization Using sklearn

To standardize your data, you need to import the StandardScaler from the sklearn library and apply it to our dataset. Here's how you can do it:

```
# data standardization with sklearn

from sklearn.preprocessing import StandardScaler

# copy of datasets

X_train_stand = X_train.copy()

X_test_stand = X_test.copy()

# numerical features

num_cols =
['Item_Weight', 'Item_Visibility', 'Item_MRP', 'Outlet_Establishment_Year']

# apply standardization on numerical features

for i in num_cols:

    # fit on training data column
```

```

scale = StandardScaler().fit(X_train_stand[[i]])

# transform the training data column

X_train_stand[i] = scale.transform(X_train_stand[[i]])

# transform the testing data column

X_test_stand[i] = scale.transform(X_test_stand[[i]])

```

You would have noticed that I only applied standardization to my numerical columns, not the other [One-Hot Encoded](#) features. Standardizing the One-Hot encoded features would mean assigning a distribution to categorical features. You don't want to do that!

But why did I not do the same while normalizing the data? Because One-Hot encoded features are already in the range between 0 to 1. So, normalization would not affect their value.

Right, let's have a look at how standardization has transformed our data:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Tier 2	Tier 3	Supermarket Type1	Supermarket Type2
count	6.818000e+03	6818.000000	6.818000e+03	6.818000e+03	6.818000e+03	6818.000000	6818.000000	6818.000000	6818.000000	6818.000
mean	1.704754e-16	0.355676	2.342737e-16	1.233002e-16	2.051747e-17	0.830302	0.326049	0.395571	0.651071	0.1111
std	1.000073e+00	0.478753	1.000073e+00	1.000073e+00	1.000073e+00	0.596352	0.468800	0.489009	0.476667	0.3141
min	-1.356094e+00	0.000000	-3.402972e+00	-1.759459e+00	-1.329746e+00	0.000000	0.000000	0.000000	0.000000	0.000
25%	-8.351767e-01	0.000000	-5.654603e-01	-7.589239e-01	-7.337084e-01	0.000000	0.000000	0.000000	0.000000	0.000
50%	5.250371e-03	0.000000	9.843444e-02	2.728679e-02	-1.376709e-01	1.000000	0.000000	0.000000	1.000000	0.000
75%	7.475728e-01	1.000000	7.696595e-01	7.091011e-01	1.292819e+00	1.000000	1.000000	1.000000	1.000000	0.000
max	2.011410e+00	1.000000	2.308151e+00	2.036689e+00	1.531234e+00	2.000000	1.000000	1.000000	1.000000	1.000

The numerical features are now centered on the mean with a unit standard deviation. Awesome!

Comparing Unscaled, Normalized, and Standardized Data

It is always great to visualize your data to understand the distribution present. We can see the comparison between our unscaled and scaled data using boxplots.

You can learn more about data visualization [here](#).

Combining all the steps:

Now, in the end, we can combine all the steps together to make our complete code more understandable.

```
1. # importing libraries
2. import numpy as np
3. import matplotlib.pyplot as plt
4. import pandas as pd
5. #importing datasets
6. data_set= pd.read_csv('Dataset.csv')
7. #Extracting Independent Variable
8. x= data_set.iloc[:, :-1].values
9. #Extracting Dependent variable
10. y= data_set.iloc[:, 3].values
11. #handling missing data(Replacing missing data with the mean value)
12. from sklearn.preprocessing import Imputer
13. imputer= Imputer(missing_values = 'NaN', strategy= 'mean', axis = 0)
14. #Fitting imputer object to the independent variables x.
15. imputer.fit(x[:, 1:3])
16. #Replacing missing data with the calculated mean value
17. x[:, 1:3]= imputer.transform(x[:, 1:3])
18. #for Country Variable
19. from sklearn.preprocessing import LabelEncoder, OneHotEncoder
20. label_encoder_x= LabelEncoder()
21. x[:, 0]= label_encoder_x.fit_transform(x[:, 0])
22. #Encoding for dummy variables
23. onehot_encoder= OneHotEncoder(categorical_features= [0])
24. x= onehot_encoder.fit_transform(x).toarray()
25. #encoding for purchased variable
26. label_encoder_y= LabelEncoder()
27. y= label_encoder_y.fit_transform(y)
28. # Splitting the dataset into training and test set.
29. from sklearn.model_selection import train_test_split
```

```

30. x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.2, random_state=0)
31. #Feature Scaling of datasets
32. from sklearn.preprocessing import StandardScaler
33. st_x= StandardScaler()
34. x_train= st_x.fit_transform(x_train)
35. x_test= st_x.transform(x_test)

```

What is Predictive Modeling: Predictive modeling is a probabilistic process that allows us to forecast outcomes, on the basis of some predictors. These predictors are basically features that come into play when deciding the final result, i.e. the outcome of the model.

Dimensionality reduction is the process of reducing the number of features (or dimensions) in a dataset while retaining as much information as possible. This can be done for a variety of reasons, such as to reduce the complexity of a model, to improve the performance of a learning algorithm, or to make it easier to visualize the data. There are several techniques for dimensionality reduction, including principal component analysis (PCA), singular value decomposition (SVD), and linear discriminant analysis (LDA). Each technique uses a different method to project the data onto a lower-dimensional space while preserving important information.

What is Dimensionality Reduction?

In machine learning classification problems, there are often too many factors on the basis of which the final classification is done. These factors are basically variables called features. The higher the number of features, the harder it gets to visualize the training set and then work on it. Sometimes, most of these features are correlated, and hence redundant. This is where dimensionality reduction algorithms come into play. Dimensionality reduction is the process of reducing the number of random variables under consideration, by obtaining a set of principal variables. It can be divided into feature selection and feature extraction.

Why is Dimensionality Reduction important in Machine Learning and Predictive Modeling?

An intuitive example of dimensionality reduction can be discussed through a simple e-mail classification problem, where we need to classify whether the e-mail is spam or not. This can involve a large number of features, such as whether or not the e-mail has a generic title, the content of the e-mail, whether the e-mail uses a template, etc. However, some of these features may overlap. In another condition, a classification problem that relies on both humidity and rainfall can be collapsed into just one underlying feature, since both of the aforementioned are correlated to a high degree. Hence, we can reduce the number of features in such problems. A 3-D classification problem can be hard to visualize, whereas a 2-D one can be mapped to a simple 2 dimensional space, and a 1-D problem to a simple line. The below figure illustrates this concept, where a 3-D feature space is split into two 2-D feature spaces, and later, if found to be correlated, the number of features can be reduced even further.



Components of Dimensionality Reduction

There are two components of dimensionality reduction:

- **Feature selection:** In this, we try to find a subset of the original set of variables, or features, to get a smaller subset which can be used to model the problem. It usually involves three ways:
 1. Filter
 2. Wrapper
 3. Embedded
- **Feature extraction:** This reduces the data in a high dimensional space to a lower dimension space, i.e. a space with lesser no. of dimensions.

Methods of Dimensionality Reduction

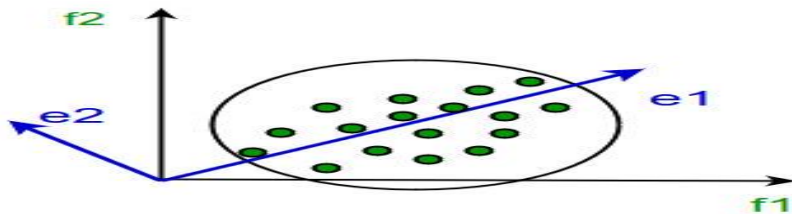
The various methods used for dimensionality reduction include:

- Principal Component Analysis (PCA)
- Linear Discriminant Analysis (LDA)
- Generalized Discriminant Analysis (GDA)

Dimensionality reduction may be both linear or non-linear, depending upon the method used. The prime linear method, called Principal Component Analysis, or PCA, is discussed below.

Principal Component Analysis

This method was introduced by Karl Pearson. It works on a condition that while the data in a higher dimensional space is mapped to data in a lower dimension space, the variance of the data in the lower dimensional space should be maximum.



It involves the following steps:

- Construct the covariance matrix of the data.
- Compute the eigenvectors of this matrix.
- Eigenvectors corresponding to the largest eigenvalues are used to reconstruct a large fraction of variance of the original data.

Hence, we are left with a lesser number of eigenvectors, and there might have been some data loss in the process. But, the most important variances should be retained by the remaining eigenvectors.

Advantages of Dimensionality Reduction

- It helps in data compression, and hence reduced storage space.
- It reduces computation time.
- It also helps remove redundant features, if any.

Disadvantages of Dimensionality Reduction

- It may lead to some amount of data loss.
- PCA tends to find linear correlations between variables, which is sometimes undesirable.
- PCA fails in cases where mean and covariance are not enough to define datasets.
- We may not know how many principal components to keep- in practice, some thumb rules are applied.

Important points:

- Dimensionality reduction is the process of reducing the number of features in a dataset while retaining as much information as possible. This can be done to reduce the complexity of a model, improve the performance of a learning algorithm, or make it easier to visualize the data.
- Techniques for dimensionality reduction include: principal component analysis (PCA), singular value decomposition (SVD), and linear discriminant analysis (LDA).

- Each technique projects the data onto a lower-dimensional space while preserving important information.
- Dimensionality reduction is performed during pre-processing stage before building a model to improve the performance
- It is important to note that dimensionality reduction can also discard useful information, so care must be taken when applying these techniques.

UNIT-03

Regression analysis is a statistical method to model the relationship between a dependent (target) and independent (predictor) variables with one or more independent variables. More specifically, 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.

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:

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

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.

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. In simple words, **"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.

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?

As mentioned above, 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:

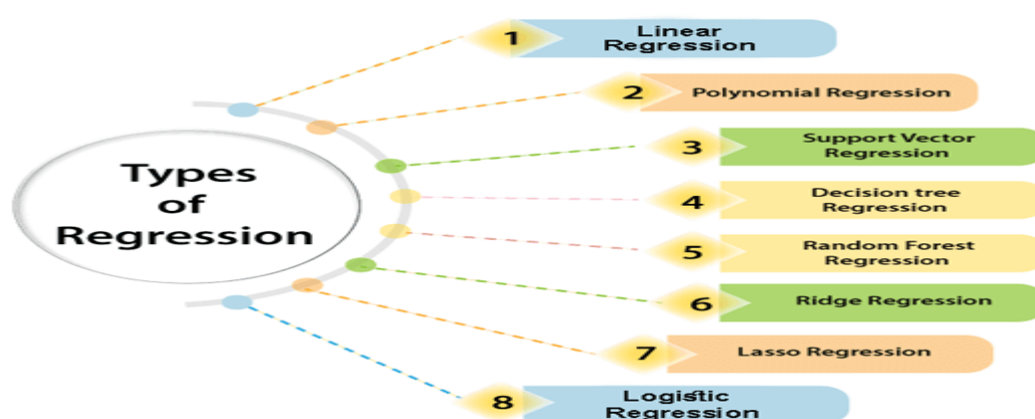
- Regression estimates the relationship between the target and the independent variable.
- It is used to find the trends in data.

- 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:

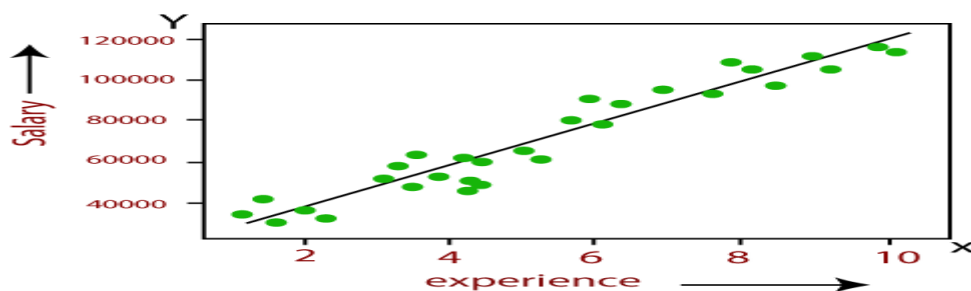
- **Linear Regression**
- **Logistic Regression**
- **Polynomial Regression**
- **Support Vector Regression**
- **Decision Tree Regression**
- **Random Forest Regression**



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:

$$1. Y = aX + b$$

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.**

Logistic Regression:

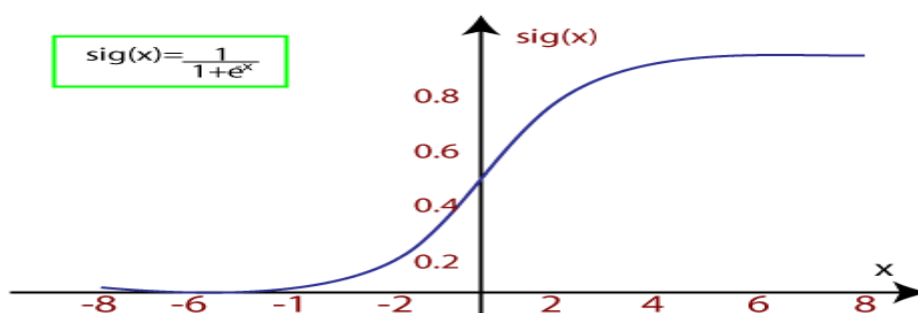
- Logistic regression is another supervised learning algorithm which is used to solve the classification problems. In **classification problems**, we have dependent variables in a binary or discrete format such as 0 or 1.

- 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.

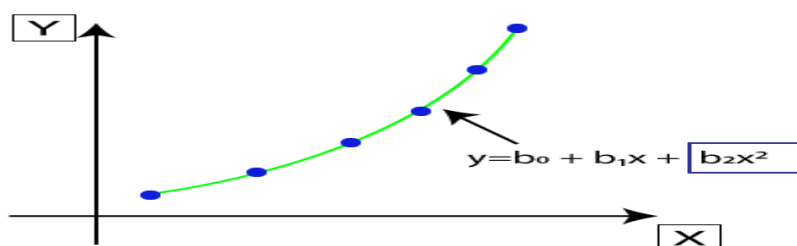
There are three types of logistic regression:

- **Binary(0/1, pass/fail)**
- **Multi(cats, dogs, lions)**
- **Ordinal(low, medium, high)**

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

Note: This is different from Multiple Linear regression in such a way that in Polynomial regression, a single element has different degrees instead of multiple variables with the same degree.

Support Vector Regression:

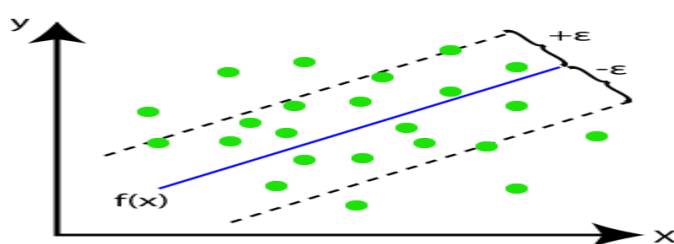
Support Vector Machine is a supervised learning algorithm which can be used for regression as well as classification problems. So if we use it for regression problems, then it is termed as Support Vector Regression.

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.

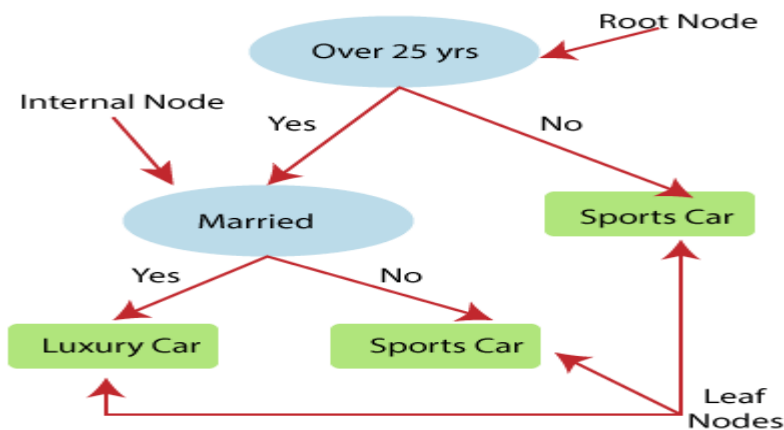
In SVR, we always try to determine a hyperplane with a maximum margin, so that maximum number of datapoints are covered in that margin. ***The main goal of SVR is to consider the maximum datapoints within the boundary lines and the hyperplane (best-fit line) must contain a maximum number of datapoints.***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



Here, the blue line is called hyperplane, and the other two lines are known as boundary lines.

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. Consider the below image:



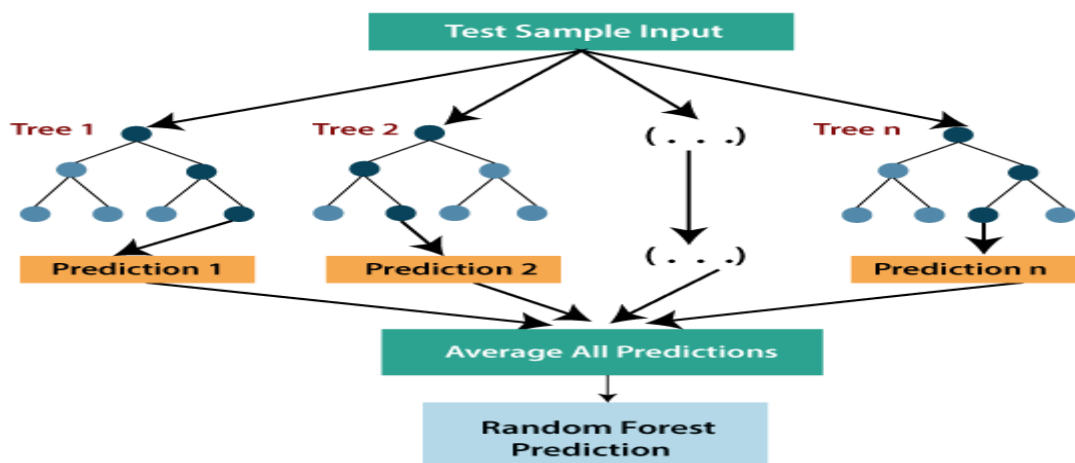
Above image showing the example of Decision Tree regression, here, the model is trying to predict the choice of a person between Sports cars or Luxury car.

Random forest regression

- 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.

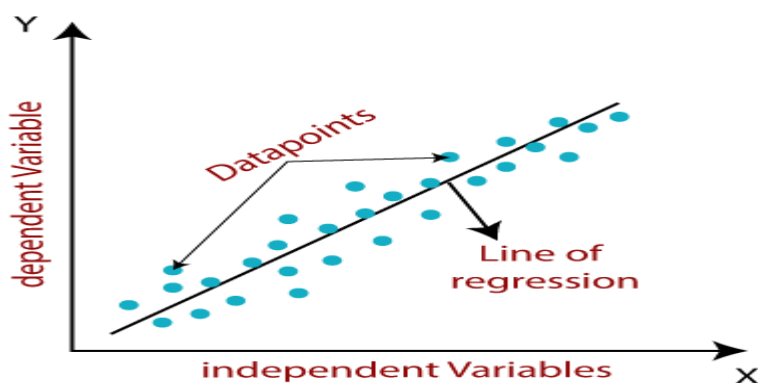


Linear Regression in Machine Learning

Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as **sales, salary, age, product price**, etc.

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

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

Here,

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

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

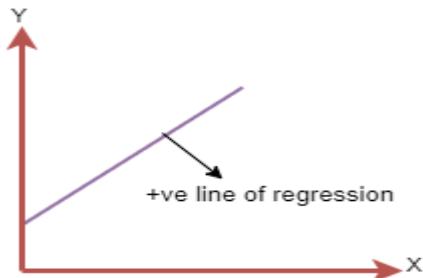
Linear regression can be further divided into two types of the algorithm:

- **Simple Linear Regression:**
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.
- **Multiple Linear regression:**
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

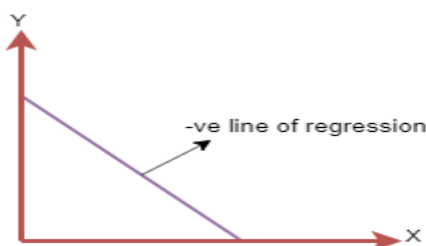
A linear line showing the relationship between the dependent and independent variables is called a **regression line**. A regression line can show two types of relationship:

- **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.



The line equation will be: $Y = a_0 + a_1X$

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



The line of equation will be: $Y = -a_0 + a_1X$

Finding the best fit line:

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

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

Cost function-

- The different values for weights or coefficient of lines (a_0 , a_1) gives the different line of regression, and the cost function is used to estimate the values of the coefficient for the best fit line.
- Cost function optimizes the regression coefficients or weights. It measures how a linear regression model is performing.
- We can use the cost function to find the accuracy of the **mapping function**, which maps the input variable to the output variable. This mapping function is also known as **Hypothesis function**.

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:

$$MSE = \frac{1}{N} \sum_{i=1}^n (y_i - (a_1 x_i + a_0))^2$$

Where,

N=Total	number	of	observation
Y_i	=	Actual	value
$(a_1 x_i + a_0)$	=	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.

Gradient Descent:

- Gradient descent is used to minimize the MSE by calculating the gradient of the cost function.
- A regression model uses gradient descent to update the coefficients of the line by reducing the cost function.
- It is done by a random selection of values of coefficient and then iteratively update the values to reach the minimum cost function.

Model Performance:

The Goodness of fit determines how the line of regression fits the set of observations. The process of finding the best model out of various models is called **optimization**. It can be achieved by below method:

Assumptions of Linear Regression

Below are some important assumptions of Linear Regression. These are some formal checks while building a Linear Regression model, which ensures to get the best possible result from the given dataset.

- **Linear relationship between the features and target:**
Linear regression assumes the linear relationship between the dependent and independent variables.
- **Small or no multicollinearity between the features:**
Multicollinearity means high-correlation between the independent variables. Due to multicollinearity, it may difficult to find the true relationship between the predictors and target variables. Or we can say, it is difficult to determine which predictor variable is affecting the target variable and which is not. So, the model assumes either little or no multicollinearity between the features or independent variables.
- **Homoscedasticity Assumption:**
Homoscedasticity is a situation when the error term is the same for all the values of independent variables. With homoscedasticity, there should be no clear pattern distribution of data in the scatter plot.
- **Normal distribution of error terms:**
Linear regression assumes that the error term should follow the normal distribution pattern. If error terms are not normally distributed, then confidence intervals will become either too wide or too narrow, which may cause difficulties in finding coefficients.
It can be checked using the **q-q plot**. If the plot shows a straight line without any deviation, which means the error is normally distributed.
- **No autocorrelations:**
The linear regression model assumes no autocorrelation in error terms. If there will be any correlation in the error term, then it will drastically reduce the accuracy of the model. Autocorrelation usually occurs if there is a dependency between residual errors.

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.

Simple Linear Regression Model:

The Simple Linear Regression model can be represented using the below equation:

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

Where,

a_0 = It is the intercept of the Regression line (can be obtained putting $x=0$)

a_1 = It is the slope of the regression line, which tells whether the line is increasing or decreasing.

ε = The error term. (For a good model it will be negligible)

Implementation of Simple Linear Regression Algorithm using Python

Problem Statement example for Simple Linear Regression:

Here we are taking a dataset that has two variables: salary (dependent variable) and experience (Independent variable). The goals of this problem is:

- **We want to find out if there is any correlation between these two variables**
- **We will find the best fit line for the dataset.**
- **How the dependent variable is changing by changing the independent variable.**

In this section, we will create a Simple Linear Regression model to find out the best fitting line for representing the relationship between these two variables.

To implement the Simple Linear regression model in machine learning using Python, we need to follow the below steps:

Step-1: Data Pre-processing

The first step for creating the Simple Linear Regression model is [data pre-processing](#). We have already done it earlier in this tutorial. But there will be some changes, which are given in the below steps:

- First, we will import the three important libraries, which will help us for loading the dataset, plotting the graphs, and creating the Simple Linear Regression model.
 1. **import** numpy as nm
 2. **import** matplotlib.pyplot as mtp
 3. **import** pandas as pd
- Next, we will load the dataset into our code:
 1. `data_set= pd.read_csv('Salary_Data.csv')`

By executing the above line of code (ctrl+ENTER), we can read the dataset on our Spyder IDE screen by clicking on the variable explorer option.



The screenshot shows the Spyder IDE variable explorer window. The title bar reads 'data_set - DataFrame'. The window displays a table with three columns: 'Index', 'YearsExperience', and 'Salary'. The data is as follows:

Index	YearsExperience	Salary
0	1	32383
1	1.1	45207
2	1.3	39751
3	2	43525
4	2.2	39891
5	2.7	56642
6	3	60150
7	3.2	54445
8	3.2	64445
9	3.7	57189
10	3.9	63218
11	4	55794
12	4	56957
13	4.1	57081

At the bottom of the window, there are buttons for 'Format', 'Resize', 'Background color', 'Column min/max', 'Save and Close', and 'Close'.

The above output shows the dataset, which has two variables: Salary and Experience.

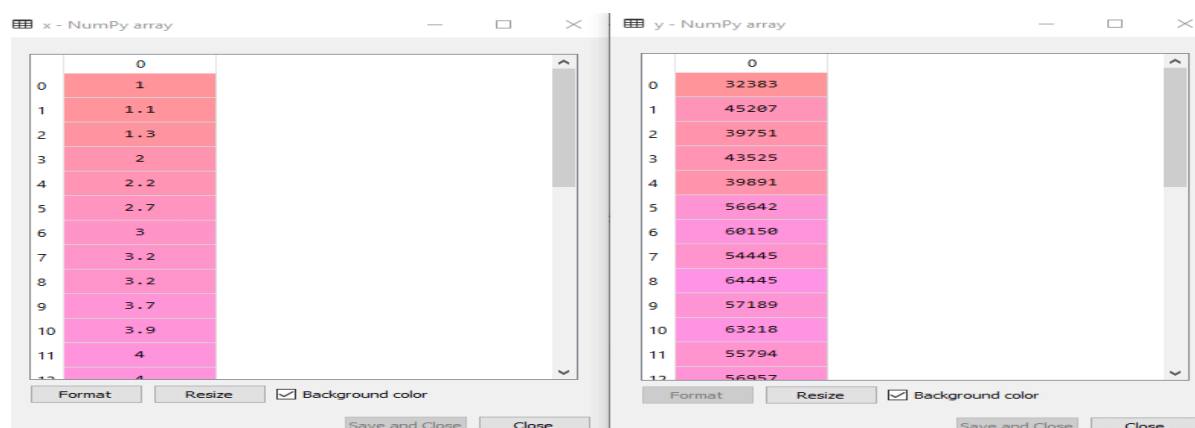
Note: In Spyder IDE, the folder containing the code file must be saved as a working directory, and the dataset or csv file should be in the same folder.

- After that, we need to extract the dependent and independent variables from the given dataset. The independent variable is years of experience, and the dependent variable is salary. Below is code for it:

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

In the above lines of code, for x variable, we have taken -1 value since we want to remove the last column from the dataset. For y variable, we have taken 1 value as a parameter, since we want to extract the second column and indexing starts from the zero.

By executing the above line of code, we will get the output for X and Y variable as:



In the above output image, we can see the X (independent) variable and Y (dependent) variable has been extracted from the given dataset.

- Next, we will 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. We are splitting our dataset so that we can train our model using a training dataset and then test the model using a test dataset. The code for this is given below:

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

By executing the above code, we will get x-test, x-train and y-test, y-train dataset. Consider the below images:

Test-dataset:

	0
0	1.3
1	10.3
2	4.1
3	3.9
4	9.5
5	8.7
6	9.6
7	4
8	5.3
9	7.9

	0
0	39751
1	122391
2	57081
3	63218
4	116969
5	109431
6	112635
7	55794
8	83088
9	101302

Training Dataset:

	0
0	2.7
1	5.1
2	3.2
3	4.5
4	8.2
5	6.8
6	1.1
7	10.5
8	3
9	2.2
10	5.8
11	6
12	3.7

	0
0	56642
1	66029
2	64445
3	61111
4	113812
5	91738
6	45207
7	121872
8	60150
9	39891
10	81363
11	93940
12	57189

- For simple linear Regression, we will not use Feature Scaling. Because Python libraries take care of it for some cases, so we don't need to perform it here. Now, our dataset is well prepared to work on it and we are going to start building a Simple Linear Regression model for the given problem.

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

Now the second step is 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**. The code for this is given below:

1. #Fitting the Simple Linear Regression model to the training dataset
2. from sklearn.linear_model **import** LinearRegression
3. regressor= LinearRegression()

4. regressor.fit(x_train, y_train)

In the above code, we have used 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.

Output:

```
Out[7]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
normalize=False)
```

Step: 3. Prediction of test set result:

dependent (salary) and an independent variable (Experience). So, now, 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.

1. #Prediction of Test and Training set result
2. y_pred= regressor.predict(x_test)
3. x_pred= regressor.predict(x_train)

On executing the above lines of code, two variables named y_pred and x_pred will generate in the variable explorer options that contain salary predictions for the training set and test set.

Output:

You can check the variable by clicking on the variable explorer option in the IDE, and also compare the result by comparing values from y_pred and y_test. By comparing these values, we can check how good our model is performing.

Step: 4. visualizing the Training set results:

Now in this step, we will visualize the training set result. To do so, we will 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, we will plot the Years of Experience of employees and on the y-axis, salary of employees. In the 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.

Now, we need to plot the regression line, so for this, we will use the **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.

Next, we will give the title for the plot. So here, we will use the **title()** function of the **pyplot** library and pass the name ("Salary vs Experience (Training Dataset)").

After that, we will assign labels for x-axis and y-axis using **xlabel()** and **ylabel()** function.

Finally, we will represent all above things in a graph using `show()`. The code is given below:

1. `mtp.scatter(x_train, y_train, color="green")`
2. `mtp.plot(x_train, x_pred, color="red")`
3. `mtp.title("Salary vs Experience (Training Dataset)")`
4. `mtp.xlabel("Years of Experience")`
5. `mtp.ylabel("Salary(In Rupees)")`
6. `mtp.show()`

Output:

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.

1. `#visualizing the Test set results`
2. `mtp.scatter(x_test, y_test, color="blue")`
3. `mtp.plot(x_train, x_pred, color="red")`
4. `mtp.title("Salary vs Experience (Test Dataset)")`
5. `mtp.xlabel("Years of Experience")`
6. `mtp.ylabel("Salary(In Rupees)")`
7. `mtp.show()`

Output:

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



In the above 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.

Multiple Linear Regression

In the previous topic, we have learned about Simple Linear Regression, where a single Independent/Predictor(X) variable is used to model the response variable (Y). But there may be various cases in which the response variable is affected by more than one predictor variable; for such cases, the Multiple Linear Regression algorithm is used.

Moreover, 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 CO₂ 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:

$$1. Y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \dots + b_nx_n \quad \text{..... (a)}$$

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:

1. **Data Pre-processing Steps**
2. **Fitting the MLR model to the training set**
3. **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:
 1. # importing libraries
 2. **import** numpy as nm
 3. **import** matplotlib.pyplot as mtp
 4. **import** pandas as pd

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

1. #importing datasets
2. data_set= pd.read_csv('50_CompList.csv')

Output: We will get the dataset as:

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

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:**

1. #Extracting Independent and dependent Variable
2. x= data_set.iloc[:, :-1].values
3. y= data_set.iloc[:, 4].values

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:

1. #Categorical data
2. from sklearn.preprocessing **import** LabelEncoder, OneHotEncoder
3. labelencoder_x= LabelEncoder()
4. x[:, 3]= labelencoder_x.fit_transform(x[:,3])
5. onehotencoder= OneHotEncoder(categorical_features= [3])
6. x= onehotencoder.fit_transform(x).toarray()

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

Output:

	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

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:

1. #avoiding the dummy variable trap:
2. 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.

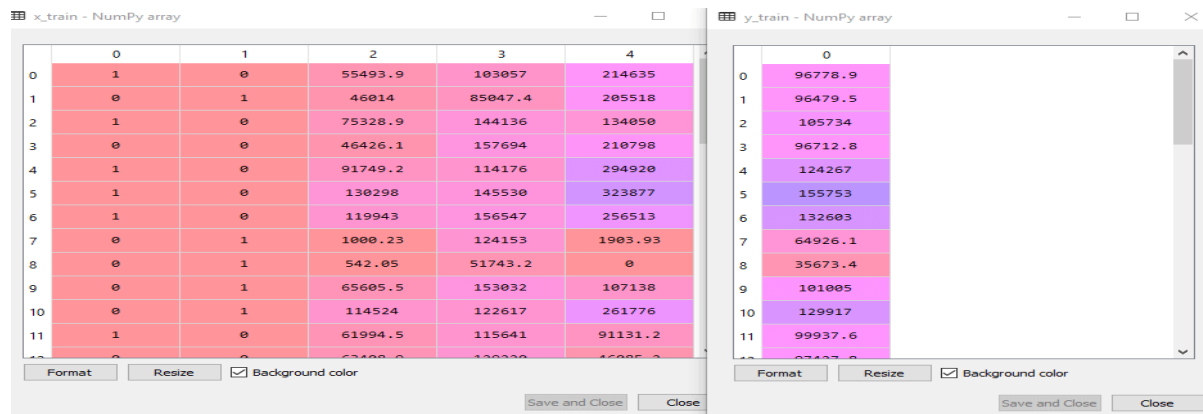
Output: The above code will split the dataset into training set and test set. You can check the output by clicking on the variable explorer option given in Spyder IDE. The test set and training set will look like the below image:

Test set:

y_test - NumPy array		x_test - NumPy array					
	0		0	1	2	3	4
0	103282		0	0	66051.5	182646	118148
1	144259		0	0	100672	91790.6	249745
2	146122		1	0	101913	110594	229161
3	77798.8		1	0	27892.9	84710.8	164471
4	191050		1	0	153442	101146	407935
5	105008		0	1	72107.6	127865	353184
6	81229.1		0	1	20229.6	65947.9	185265
7	97483.6		0	1	61136.4	152702	88218.2
8	110352		1	0	73994.6	122783	303319
9	166188		1	0	142107	91391.8	366168

Format Resize ☒ Background color Save and Close Close

Training set:



	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

	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

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:

1. #Fitting the MLR model to the training set:
2. from sklearn.linear_model **import** LinearRegression
3. regressor= LinearRegression()
4. 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.

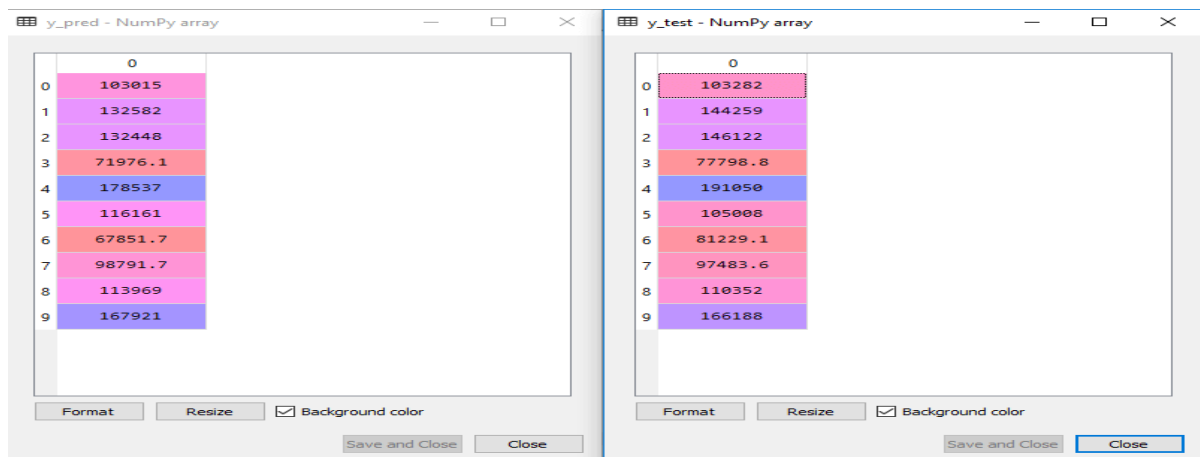
Step: 3- Prediction of Test set results:

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:

1. #Predicting the Test set result;
2. 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:



The image shows two side-by-side windows from a variable explorer, both titled 'NumPy array'. The left window is labeled 'y_pred' and the right is labeled 'y_test'. Both windows display a single column of 10 values, indexed from 0 to 9. The values are color-coded: odd indices (1, 3, 5, 7, 9) are pink, and even indices (0, 2, 4, 6, 8) are blue. The 'y_pred' window has a 'Background color' checkbox checked, while the 'y_test' window has it unchecked.

Index	y_pred	y_test
0	103015	103282
1	132582	144259
2	132448	146122
3	71976.1	77798.8
4	178537	191050
5	116161	105008
6	67851.7	81229.1
7	98791.7	97483.6
8	113969	110352
9	167921	166188

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:

1. `print('Train Score: ', regressor.score(x_train, y_train))`
2. `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:

There are mainly two applications of Multiple Linear Regression:

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

What is the Classification Algorithm?

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.

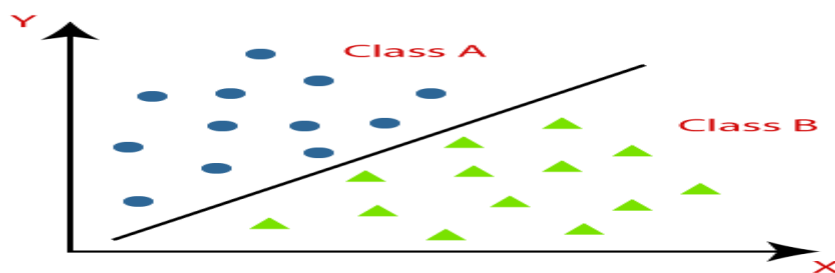
In classification algorithm, a discrete output function(y) is mapped to input variable(x).

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 it is 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.

Logistic Regression in Machine Learning

- Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.
- Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, **it gives the probabilistic values which lie between 0 and 1.**

- Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas **Logistic regression is used for solving the classification problems.**
- In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).
- The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.
- Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.
- Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification. The below image is showing the logistic function:

Note: Logistic regression uses the concept of predictive modeling as regression; therefore, it is called logistic regression, but is used to classify samples; Therefore, it falls under the classification algorithm.

Logistic Function (Sigmoid Function):

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.
- It maps any real value into another value within a range of 0 and 1.
- The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.
- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

Assumptions for Logistic Regression:

- The dependent variable must be categorical in nature.
- The independent variable should not have multi-collinearity.

Logistic Regression Equation:

The Logistic regression equation can be obtained from the Linear Regression equation. The mathematical steps to get Logistic Regression equations are given below:

- We know the equation of the straight line can be written as:

$$y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \dots + b_nx_n$$

- In Logistic Regression y can be between 0 and 1 only, so for this let's divide the above equation by (1-y):

$$\frac{y}{1-y}; 0 \text{ for } y=0, \text{ and infinity for } y=1$$

- But we need range between -[infinity] to +[infinity], then take logarithm of the equation it will become:

$$\log \left[\frac{y}{1-y} \right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \dots + b_nx_n$$

The above equation is the final equation for Logistic Regression.

Type of Logistic Regression:

On the basis of the categories, Logistic Regression can be classified into three types:

- **Binomial:** In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.
- **Multinomial:** In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"
- **Ordinal:** In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".

Python Implementation of Logistic Regression (Binomial)

To understand the implementation of Logistic Regression in Python, we will use the below example:

Example: There is a dataset given which contains the information of various users obtained from the social networking sites. There is a car making company that has

recently launched a new SUV car. So the company wanted to check how many users from the dataset, wants to purchase the car.

User ID	Gender	Age	EstimatedSalary	Purchased
15624510	Male	19	15000	0
15810944	Male	35	20000	0
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	0
15728773	Male	27	58000	0
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	0
15727311	Female	35	65000	0
15570769	Female	26	80000	0
15606274	Female	26	52000	0
15746139	Male	20	86000	0
15704987	Male	32	18000	0
15628972	Male	18	82000	0
15697686	Male	29	80000	0
15733883	Male	47	25000	1
15617482	Male	45	26000	1
15704583	Male	46	28000	1
15621083	Female	48	29000	1
15649487	Male	45	22000	1
15736760	Female	47	49000	1

Steps in Logistic Regression: To implement the Logistic Regression using Python, we will use the same steps as we have done in previous topics of Regression. Below are the steps:

- Data Pre-processing step
- Fitting Logistic Regression 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: In this step, we will pre-process/prepare the data so that we can use it in our code efficiently. It will be the same as we have done in Data pre-processing topic. The code for this is given below:

1. #Data Pre-procesing Step
2. # importing libraries
3. **import** numpy as nm
4. **import** matplotlib.pyplot as mtp
5. **import** pandas as pd
- 6.
7. #importing datasets
8. data_set= pd.read_csv('user_data.csv')

By executing the above lines of code, we will get the dataset as the output. Consider the given image:

Index	User ID	Gender	Age	EstimatedSalary	Purchased
92	15809823	Male	26	15000	0
150	15679651	Female	26	15000	0
43	15792008	Male	30	15000	0
155	15610140	Female	31	15000	0
32	15573452	Female	21	16000	0
180	15685576	Male	26	16000	0
79	15655123	Female	26	17000	0
40	15764419	Female	27	17000	0
128	15722758	Male	30	17000	0
58	15642885	Male	22	18000	0
29	15669656	Male	31	18000	0
13	15704987	Male	32	18000	0
74	15592877	Male	32	18000	0
0	15624510	Male	19	19000	0

Now, we will extract the dependent and independent variables from the given dataset. Below is the code for it:

1. #Extracting Independent and dependent Variable
2. `x= data_set.iloc[:, [2,3]].values`
3. `y= data_set.iloc[:, 4].values`

In the above code, we have taken [2, 3] for x because our independent variables are age and salary, which are at index 2, 3. And we have taken 4 for y variable because our dependent variable is at index 4. The output will be:

	0	1
0	19	19000
1	35	20000
2	26	43000
3	27	57000
4	19	76000
5	27	58000
6	27	84000
7	32	150000
8	25	33000
9	35	65000
10	26	80000
11	26	52000
12	20	86000

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0
10	0
11	0
12	0

Now we will split the dataset into a training set and test set. Below is the code for it:

1. # Splitting the dataset into training and test set.
2. `from sklearn.model_selection import train_test_split`

3. `x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)`

The output for this is given below:

**For
set:**

test

	0	1
0	-0.804802	0.504964
1	-0.0125441	-0.567782
2	-0.309641	0.157046
3	-0.804802	0.273019
4	-0.309641	-0.567782
5	-1.1019	-1.43758
6	-0.70577	-1.58254
7	-0.210609	2.15757
8	-1.99319	-0.0459058
9	0.878746	-0.770734
10	-0.804802	-0.596776
11	-1.00287	-0.422817
12	-0.111576	-0.422817

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0
10	0
11	0
12	0

For training set:

	0	1
0	-0.804802	0.504964
1	-0.0125441	-0.567782
2	-0.309641	0.157046
3	-0.804802	0.273019
4	-0.309641	-0.567782
5	-1.1019	-1.43758
6	-0.70577	-1.58254
7	-0.210609	2.15757
8	-1.99319	-0.0459058
9	0.878746	-0.770734
10	-0.804802	-0.596776
11	-1.00287	-0.422817
12	-0.111576	-0.422817

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0
10	0
11	0
12	0

In logistic regression, we will do feature scaling because we want accurate result of predictions. Here we will only scale the independent variable because dependent variable have only 0 and 1 values. Below is the code for it:

1. `#feature Scaling`
2. `from sklearn.preprocessing import StandardScaler`
3. `st_x= StandardScaler()`

4. `x_train= st_x.fit_transform(x_train)`
5. `x_test= st_x.transform(x_test)`

The scaled output is given below:

	0	1
0	-0.804802	0.504964
1	-0.0125441	-0.567782
2	-0.309641	0.157046
3	-0.804802	0.273019
4	-0.309641	-0.567782
5	-1.1019	-1.43758
6	-0.70577	-1.58254
7	-0.210609	2.15757
8	-1.99319	-0.0459058
9	0.878746	-0.770734
10	-0.804802	-0.596776
11	-1.00287	-0.422817
12	-0.111576	-0.422817

	0	1
0	0.581649	-0.886707
1	-0.606738	1.46174
2	-0.0125441	-0.567782
3	-0.606738	1.89663
4	1.37391	-1.40858
5	1.47294	0.997847
6	0.0864882	-0.799728
7	-0.0125441	-0.248858
8	-0.210609	-0.567782
9	-0.210609	-0.190872
10	-0.309641	-1.29261
11	-0.309641	-0.567782
12	0.383585	0.0990599

2. Fitting Logistic Regression to the Training set:

We have well prepared our dataset, and now we will train the dataset using the training set. For providing training or fitting the model to the training set, we will import the **LogisticRegression** class of the **sklearn** library.

After importing the class, we will create a classifier object and use it to fit the model to the logistic regression. Below is the code for it:

1. `#Fitting Logistic Regression to the training set`
2. `from sklearn.linear_model import LogisticRegression`
3. `classifier= LogisticRegression(random_state=0)`
4. `classifier.fit(x_train, y_train)`

Output: By executing the above code, we will get the below output:

Out[5]:

1. `LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,`
2.
3.
4.
5.

Hence our model is well fitted to the training set.

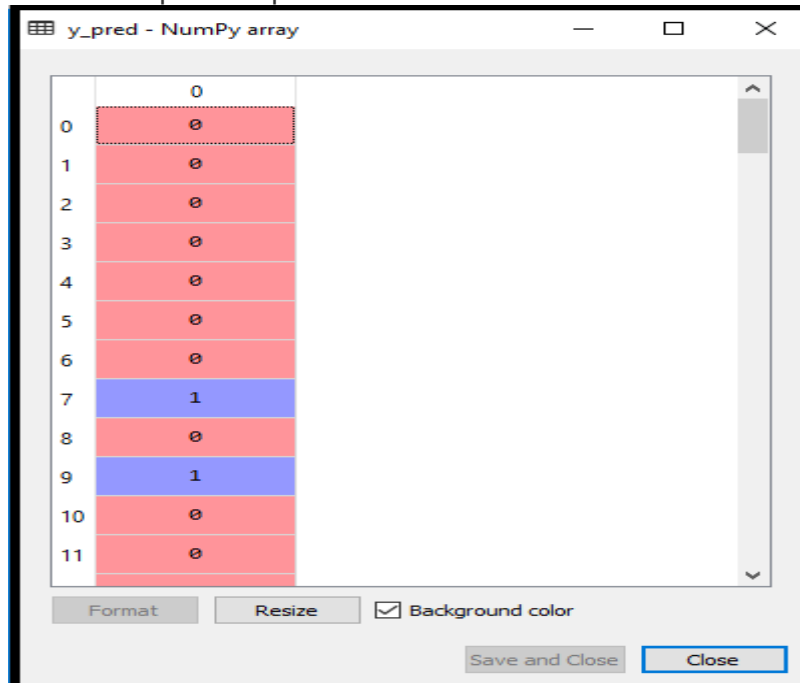
3. Predicting the Test Result

Our model is well trained on the training set, so we will now predict the result by using test set data. Below is the code for it:

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

In the above code, we have created a `y_pred` vector to predict the test set result.

Output: By executing the above code, a new vector (`y_pred`) will be created under the variable explorer option. It can be seen as:



The above output image shows the corresponding predicted users who want to purchase or not purchase the car.

4. Test Accuracy of the result

Now we will create the confusion matrix here to check the accuracy of the classification. To create it, we need to import the **confusion_matrix** function of the sklearn library. After importing the function, we will call it using a new variable **cm**. The function takes two parameters, mainly **y_true**(the actual values) and **y_pred** (the targeted value return by the classifier). Below is the code for it:

1. `#Creating the Confusion matrix`
2. `from sklearn.metrics import confusion_matrix`
3. `cm= confusion_matrix()`

Output:

By executing the above code, a new confusion matrix will be created. Consider the below image:



We can find the accuracy of the predicted result by interpreting the confusion matrix. By above output, we can interpret that $65+24= 89$ (Correct Output) and $8+3= 11$ (Incorrect Output).

5. Visualizing the training set result

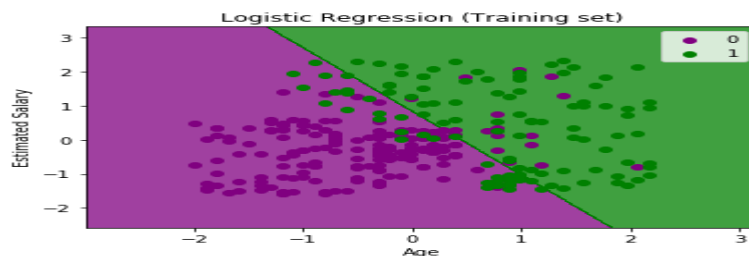
Finally, we will visualize the training set result. To visualize the result, we will use **ListedColormap** class of matplotlib library. Below is the code for it:

1. #Visualizing the training 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('Logistic Regression (Training set)')
14. mtp.xlabel('Age')
15. mtp.ylabel('Estimated Salary')
16. mtp.legend()
17. mtp.show()

In the above code, we have imported the **ListedColormap** class of Matplotlib library to create the colormap for visualizing the result. We have created two new variables **x_set** and **y_set** to replace **x_train** and **y_train**. After that, we have used the **nm.meshgrid** command to create a rectangular grid, which has a range of -1 (minimum) to 1 (maximum). The pixel points we have taken are of 0.01 resolution.

To create a filled contour, we have used **mtp.contourf** command, it will create regions of provided colors (purple and green). In this function, we have passed the **classifier.predict** to show the predicted data points predicted by the classifier.

Output: By executing the above code, we will get the below output:



The graph can be explained in the below points:

- In the above graph, we can see that there are some **Green points** within the green region and **Purple points** within the purple region.
- All these data points are the observation points from the training set, which shows the result for purchased variables.
- This graph is made by using two independent variables i.e., **Age on the x-axis** and **Estimated salary on the y-axis**.
- The **purple point observations** are for which purchased (dependent variable) is probably 0, i.e., users who did not purchase the SUV car.
- The **green point observations** are for which purchased (dependent variable) is probably 1 means user who purchased the SUV car.
- We can also estimate from the graph that the users who are younger with low salary, did not purchase the car, whereas older users with high estimated salary purchased the car.
- But there are some purple points in the green region (Buying the car) and some green points in the purple region (Not buying the car). So we can say that younger users with a high estimated salary purchased the car, whereas an older user with a low estimated salary did not purchase the car.

The goal of the classifier:

We have successfully visualized the training set result for the logistic regression, and our goal for this classification is to divide the users who purchased the SUV car and who did not purchase the car. So from the output graph, we can clearly see the two regions (Purple and Green) with the observation points. The Purple region is for those users who didn't buy the car, and Green Region is for those users who purchased the car.

Linear Classifier:

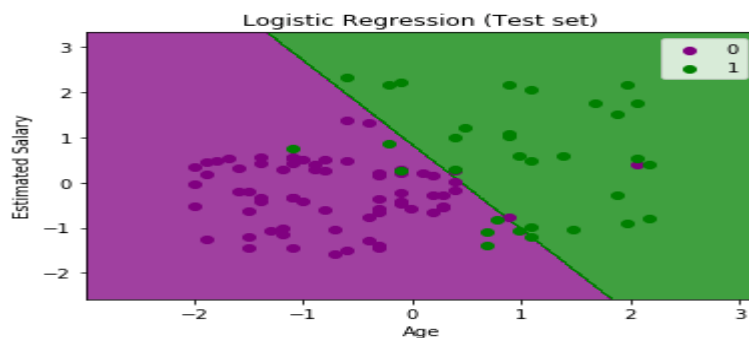
As we can see from the graph, the classifier is a Straight line or linear in nature as we have used the Linear model for Logistic Regression. In further topics, we will learn for non-linear Classifiers.

Visualizing the test set result:

Our model is well trained using the training dataset. Now, we will visualize the result for new observations (Test set). The code for the test set will remain same as above except that here we will use **x_test** and **y_test** instead of **x_train** and **y_train**. 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() + 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('Logistic Regression (Test set)')
14. mtp.xlabel('Age')
15. mtp.ylabel('Estimated Salary')
16. mtp.legend()
17. mtp.show()
```

Output:



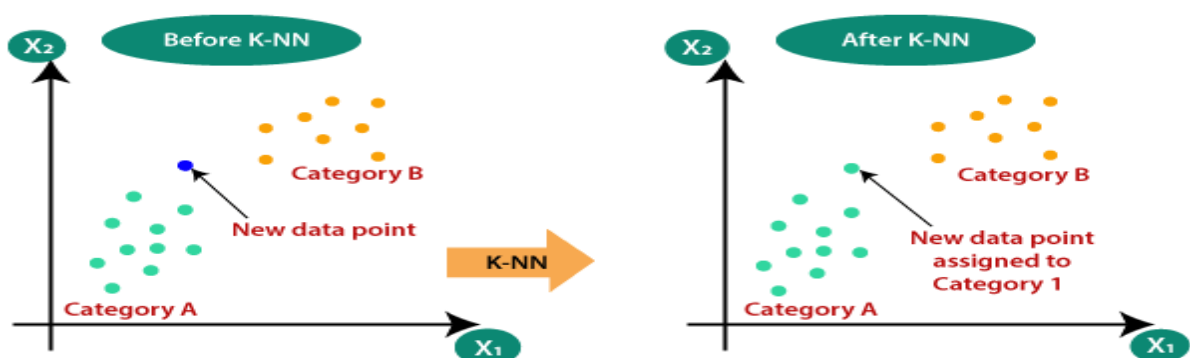
The above graph shows the test set result. As we can see, the graph is divided into two regions (Purple and Green). And Green observations are in the green region, and Purple observations are in the purple region. So we can say it is a good prediction and model. Some of the green and purple data points are in different regions, which can be ignored as we have already calculated this error using the confusion matrix (11 Incorrect output).

K-Nearest Neighbor(KNN) Algorithm for Machine Learning

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.
- **Example:** Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

Why do we need a K-NN Algorithm?

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x_1 , so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:

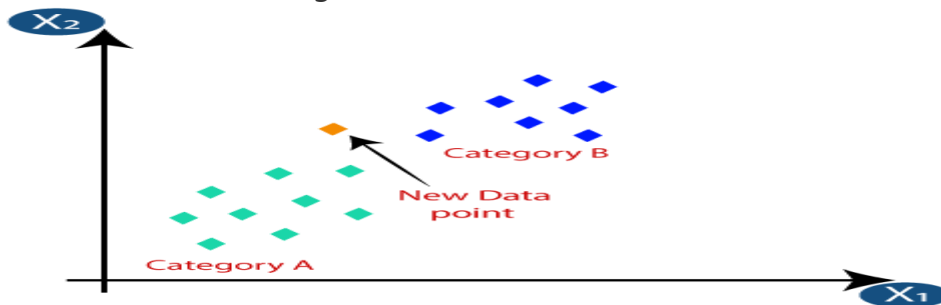


How does K-NN work?

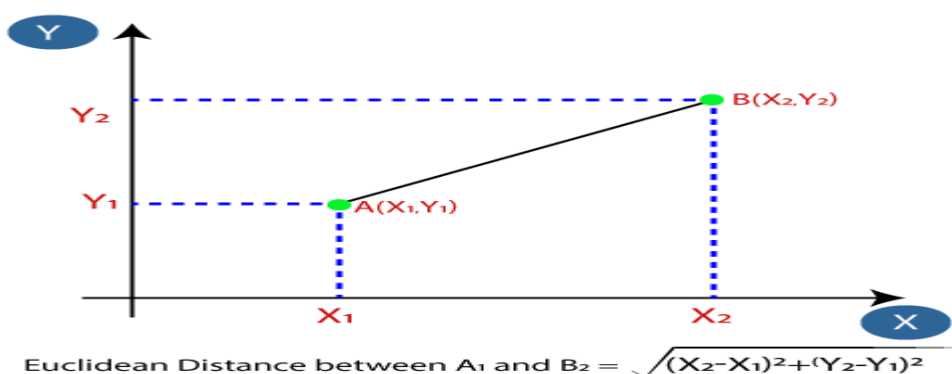
The K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

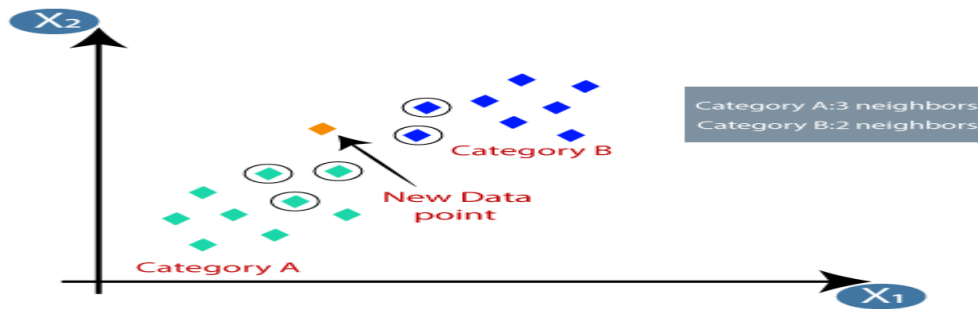
Suppose we have a new data point and we need to put it in the required category. Consider the below image:



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



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



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

How to select the value of K in the K-NN Algorithm?

Below are some points to remember while selecting the value of K in the K-NN algorithm

- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.

Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

Disadvantages of KNN Algorithm:

- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.

Python implementation of the KNN algorithm

To do the Python implementation of the K-NN algorithm, we will use the same problem and dataset which we have used in Logistic Regression. But here we will improve the performance of the model. Below is the problem description:

Problem for K-NN Algorithm: There is a Car manufacturer company that has manufactured a new SUV car. The company wants to give the ads to the users who are interested in buying that SUV. So for this problem, we have a dataset that contains multiple user's information through the social network. The dataset contains lots of information but the **Estimated Salary** and **Age** we will consider for the independent variable and the **Purchased variable** is for the dependent variable. Below is the dataset:

Steps to implement the K-NN algorithm:

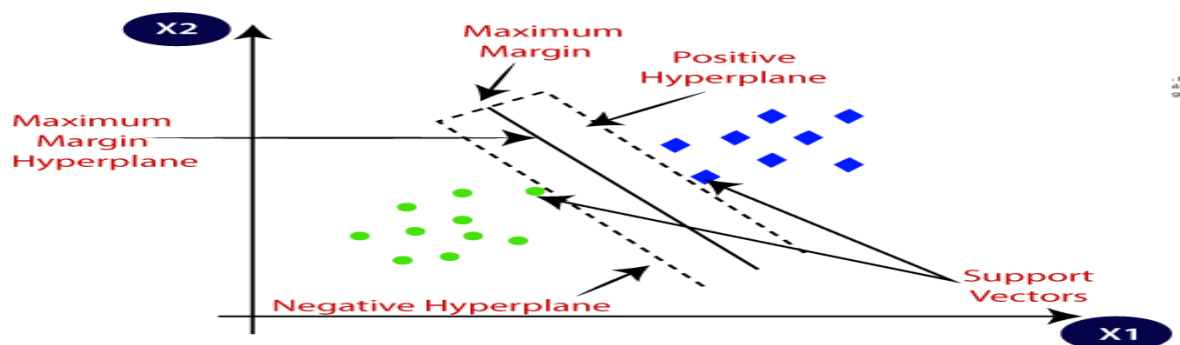
- Data Pre-processing step
- Fitting the K-NN algorithm to the Training set
- Predicting the test result
- Test accuracy of the result(Creation of Confusion matrix)
- Visualizing the test set result.

Support Vector Machine Algorithm

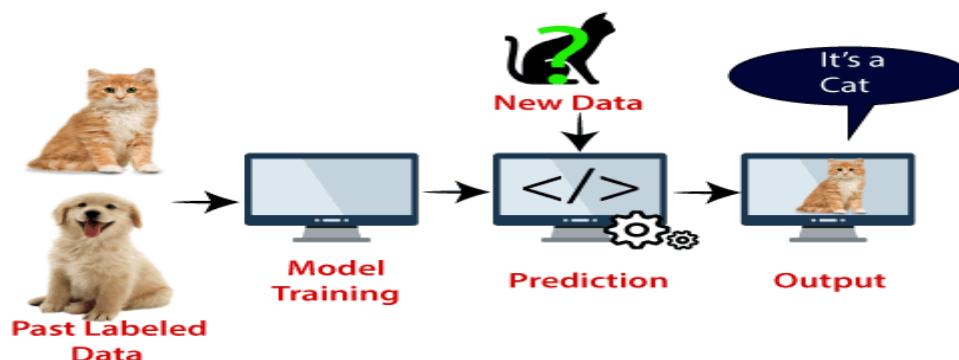
Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

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

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



Example: SVM can be understood with the example that we have used in the KNN classifier. Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm. We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature. So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case of cat and dog. On the basis of the support vectors, it will classify it as a cat. Consider the below diagram:



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

Types of SVM

SVM can be of two types:

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

Hyperplane and Support Vectors in the SVM algorithm:

Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

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

We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

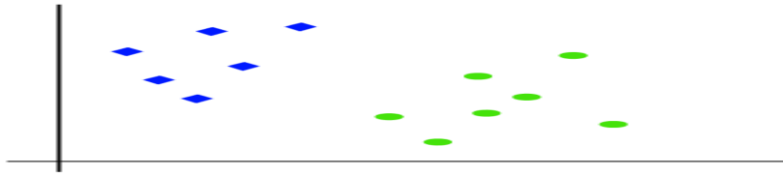
Support Vectors:

The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

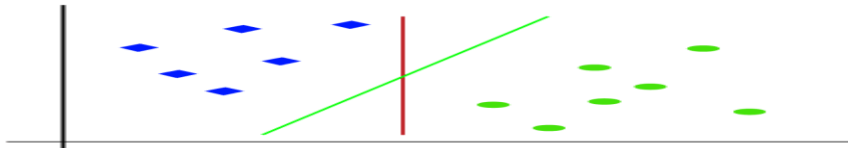
How does SVM works?

Linear SVM:

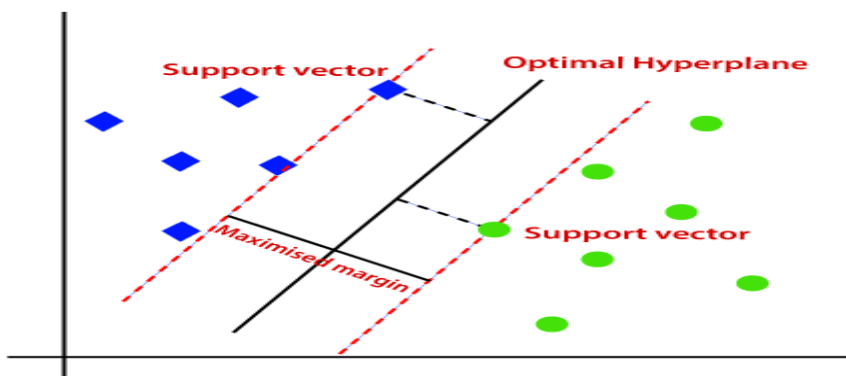
The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x_1 and x_2 . We want a classifier that can classify the pair(x_1 , x_2) of coordinates in either green or blue. Consider the below image:



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

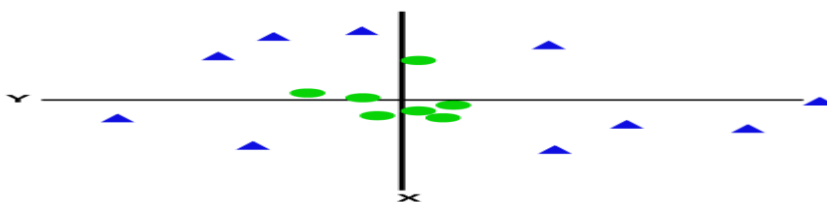


Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane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



Non-Linear SVM:

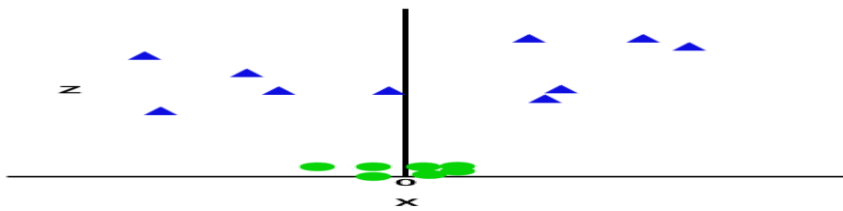
If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:



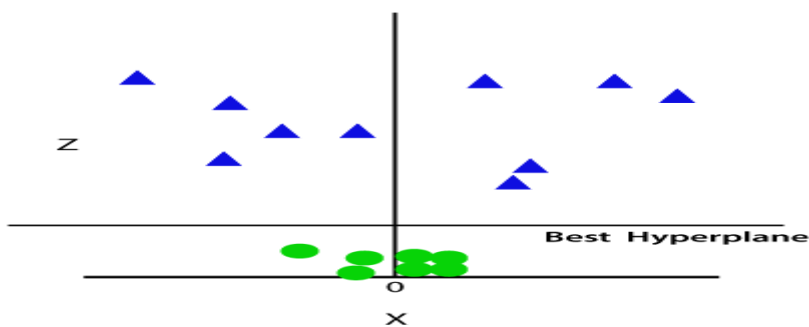
So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y , so for non-linear data, we will add a third dimension z . It can be calculated as:

$$z = x^2 + y^2$$

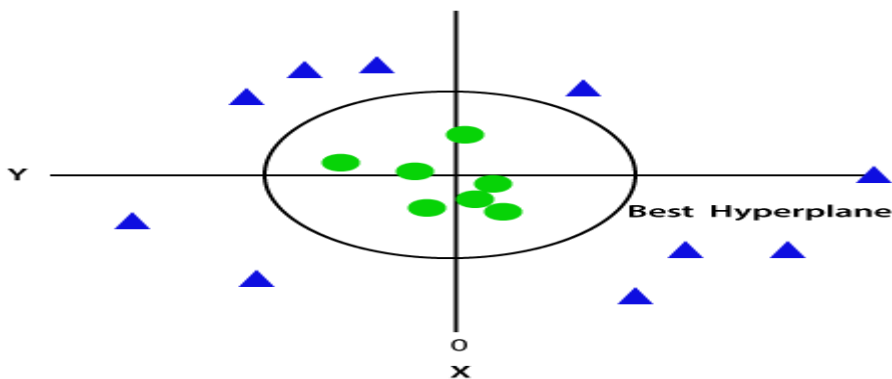
By adding the third dimension, the sample space will become as below image:



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



Since we are in 3-d Space, hence it is looking like a plane parallel to the x-axis. If we convert it in 2d space with $z=1$, then it will become as:



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

Python Implementation of Support Vector Machine

- **Data Pre-processing step**

Till the Data pre-processing step, the code will remain the same. Below is the code:

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

After executing the above code, we will pre-process the data. The code will give the dataset as:

Fitting the SVM classifier to the training set:

Now the training set will be fitted to the SVM classifier. To create the SVM classifier, we will import **SVC** class from **Sklearn.svm** library. Below is the code for it:

```
1. from sklearn.svm import SVC # "Support vector classifier"
2. classifier = SVC(kernel='linear', random_state=0)
3. classifier.fit(x_train, y_train)
```

In the above code, we have used **kernel='linear'**, as here we are creating SVM for linearly separable data. However, we can change it for non-linear data. And then we fitted the classifier to the training dataset(x_train, y_train)

Output:

```
Out[8]:
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='linear', max_iter=-1, probability=False, random_state=0,
    shrinking=True, tol=0.001, verbose=False)
```

The model performance can be altered by changing the value of **C(Regularization factor), gamma, and kernel**.

- **Predicting the test set result:**

Now, we will predict the output for test set. For this, we will create a new vector `y_pred`. Below is the code for it:

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

After getting the `y_pred` vector, we can compare the result of **y_pred** and **y_test** to check the difference between the actual value and predicted value.

Output: Below is the output for the prediction of the test set:

- **Creating the confusion matrix:**

Now we will see the performance of the SVM classifier that how many incorrect predictions are there as compared to the Logistic regression classifier. To create the confusion matrix, we need to import the **confusion_matrix** function of the sklearn library. After importing the function, we will call it using a new variable **cm**. The function takes two parameters, mainly **y_true**(the actual values) and **y_pred** (the targeted value return by the classifier). 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 output image, there are $66+24=90$ correct predictions and $8+2=10$ correct predictions. Therefore we can say that our SVM model improved as compared to the Logistic regression model.

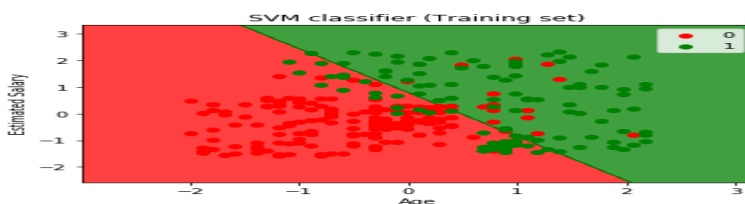
○ Visualizing the training set result:

Now we will visualize the training set result, 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(('red', '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(('red', 'green'))(i), label = j)
12. mtp.title('SVM classifier (Training set)')
13. mtp.xlabel('Age')
14. mtp.ylabel('Estimated Salary')
15. mtp.legend()
16. mtp.show()
```

Output:

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



As we can see, the above output is appearing similar to the Logistic regression output. In the output, we got the straight line as hyperplane because we have **used a linear kernel in the classifier**. And we have also discussed above that for the 2d space, the hyperplane in SVM is a straight line.

○ Visualizing the test set result:

```
1. #Visualizing 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() + 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(('red', 'green' )))
```

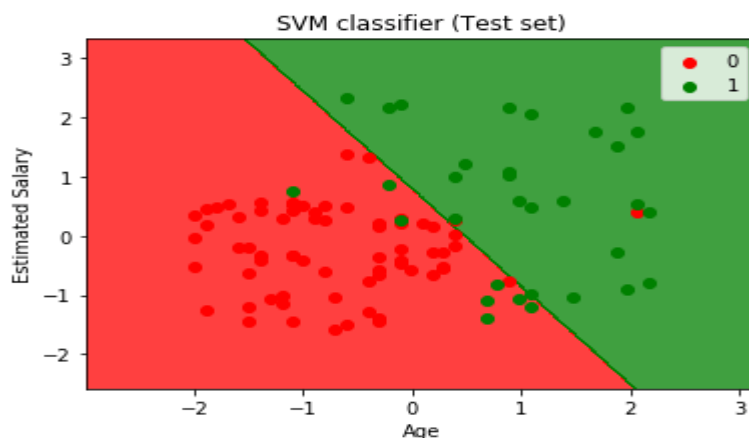
```

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(('red', 'green'))(i), label = j)
13. mtp.title('SVM classifier (Test set)')
14. mtp.xlabel('Age')
15. mtp.ylabel('Estimated Salary')
16. mtp.legend()
17. mtp.show()

```

Output:

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



As we can see in the above output image, the SVM classifier has divided the users into two regions (Purchased or Not purchased). Users who purchased the SUV are in the red region with the red scatter points. And users who did not purchase the SUV are in the green region with green scatter points. The hyperplane has divided the two classes into Purchased and not purchased variable.

Naïve Bayes Classifier Algorithm

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems.
- It is mainly used in *text classification* that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- **It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.**

- Some popular examples of Naïve Bayes Algorithm are **spam filtration, Sentimental analysis, and classifying articles.**

Why is it called Naïve Bayes?

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

- **Naïve:** It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
- **Bayes:** It is called Bayes because it depends on the principle of [Bayes' Theorem](#).

Bayes' Theorem:

- Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
- The formula for Bayes' theorem is given as:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Where,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

P(B) is Marginal Probability: Probability of Evidence.

Working of Naïve Bayes' Classifier:

Working of Naïve Bayes' Classifier can be understood with the help of the below example:

Suppose we have a dataset of **weather conditions** and corresponding target variable "**Play**". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions. So to solve this problem, we need to follow the below steps:

1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.

Problem: If the weather is sunny, then the Player should play or not?

Solution: To solve this, first consider the below dataset:

	Outlook	Play
0	Rainy	Yes
1	Sunny	Yes
2	Overcast	Yes
3	Overcast	Yes
4	Sunny	No
5	Rainy	Yes
6	Sunny	Yes
7	Overcast	Yes
8	Rainy	No
9	Sunny	No
10	Sunny	Yes
11	Rainy	No

12	Overcast	Yes
13	Overcast	Yes

Frequency table for the Weather Conditions:

Weather	Yes	No
Overcast	5	0
Rainy	2	2
Sunny	3	2
Total	10	5

Likelihood table weather condition:

Weather	No	Yes	
Overcast	0	5	$5/14 = 0.35$
Rainy	2	2	$4/14 = 0.29$
Sunny	2	3	$5/14 = 0.35$
All	$4/14 = 0.29$	$10/14 = 0.71$	

Applying Bayes' theorem:

$$P(\text{Yes}|\text{Sunny}) = P(\text{Sunny}|\text{Yes}) * P(\text{Yes}) / P(\text{Sunny})$$

$$P(\text{Sunny}|\text{Yes}) = 3/10 = 0.3$$

$$P(\text{Sunny}) = 0.35$$

$$P(\text{Yes}) = 0.71$$

$$\text{So } P(\text{Yes}|\text{Sunny}) = 0.3 * 0.71 / 0.35 = \mathbf{0.60}$$

$$P(\text{No}|\text{Sunny}) = P(\text{Sunny}|\text{No}) * P(\text{No}) / P(\text{Sunny})$$

$$P(\text{Sunny}|\text{No}) = 2/4 = 0.5$$

$$P(\text{No}) = 0.29$$

$$P(\text{Sunny}) = 0.35$$

$$\text{So } P(\text{No}|\text{Sunny}) = 0.5 * 0.29 / 0.35 = \mathbf{0.41}$$

So as we can see from the above calculation that $P(\text{Yes}|\text{Sunny}) > P(\text{No}|\text{Sunny})$

Hence on a Sunny day, Player can play the game.

Advantages of Naïve Bayes Classifier:

- Naïve Bayes is one of the fast and easy ML algorithms to predict a class of datasets.

- It can be used for Binary as well as Multi-class Classifications.
- It performs well in Multi-class predictions as compared to the other Algorithms.
- It is the most popular choice for **text classification problems**.

Disadvantages of Naïve Bayes Classifier:

- Naive Bayes assumes that all features are independent or unrelated, so it cannot learn the relationship between features.

Applications of Naïve Bayes Classifier:

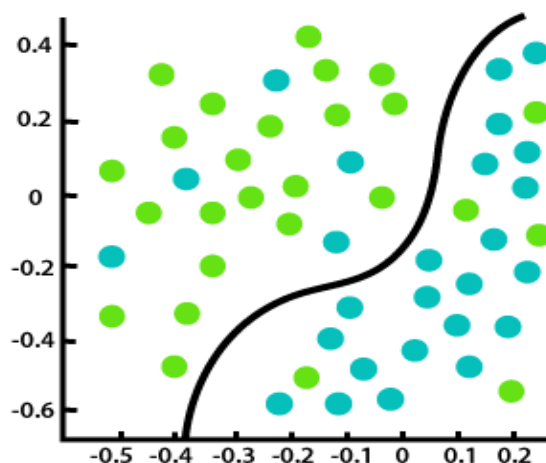
- It is used for **Credit Scoring**.
- It is used in **medical data classification**.
- It can be used in **real-time predictions** because Naïve Bayes Classifier is an eager learner.
- It is used in Text classification such as **Spam filtering** and **Sentiment analysis**.

Steps to implement:

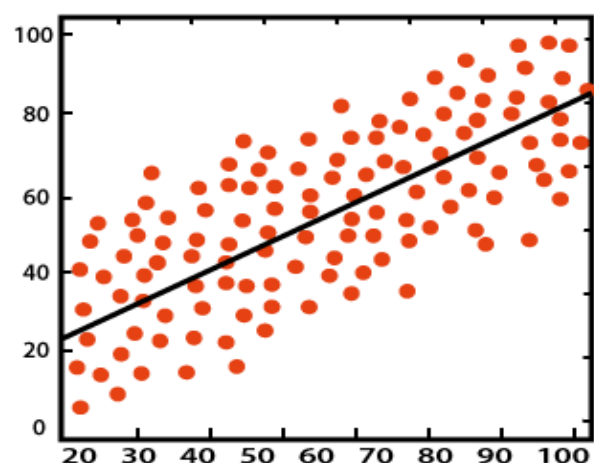
- Data Pre-processing step
- Fitting Naive Bayes to the Training set
- Predicting the test result
- Test accuracy of the result(Creation of Confusion matrix)
- Visualizing the test set result.

The main difference between Regression and Classification algorithms that Regression algorithms are used to **predict the continuous** values such as price, salary, age, etc. and Classification algorithms are used to **predict/Classify the discrete values** such as Male or Female, True or False, Spam or Not Spam, etc.

Consider the below diagram:



Classification



Regression

Difference between Regression and Classification

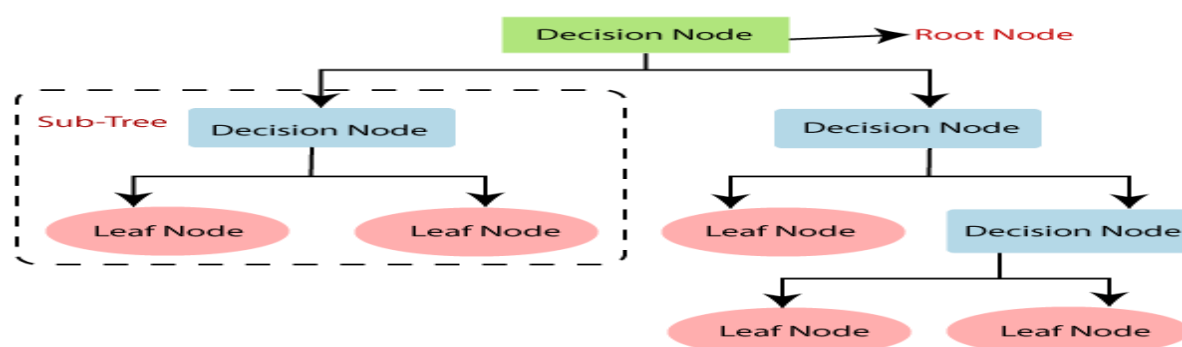
Regression Algorithm	Classification Algorithm
In Regression, the output variable must be of continuous nature or real value.	In Classification, the output variable must be a discrete value.
The task of the regression algorithm is to map the input value (x) with the continuous output variable(y).	The task of the classification algorithm is to map the input value(x) with the discrete output variable(y).
Regression Algorithms are used with continuous data.	Classification Algorithms are used with discrete data.
In Regression, we try to find the best fit line, which can predict the output more accurately.	In Classification, we try to find the decision boundary, which can divide the dataset into different classes.
Regression algorithms can be used to solve the regression problems such as Weather Prediction, House price prediction, etc.	Classification Algorithms can be used to solve classification problems such as Identification of spam emails, Speech Recognition, Identification of cancer cells, etc.
The regression Algorithm can be further divided into Linear and Non-linear Regression.	The Classification algorithms can be divided into Binary Classifier and Multi-class Classifier.

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.

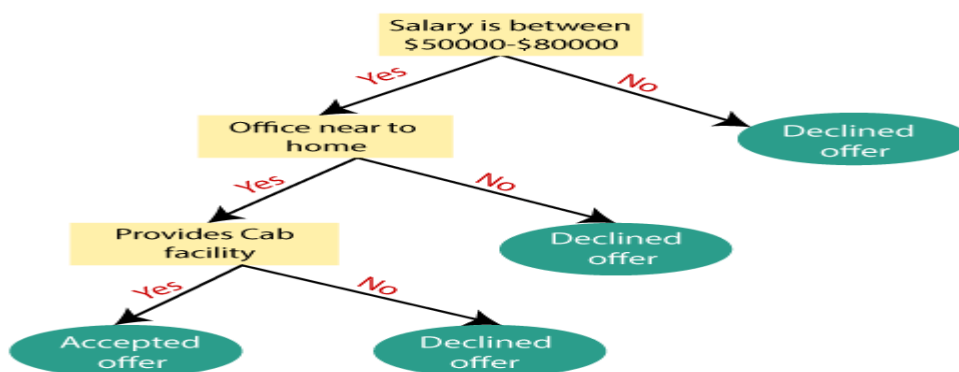
How does the Decision Tree algorithm Work?

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

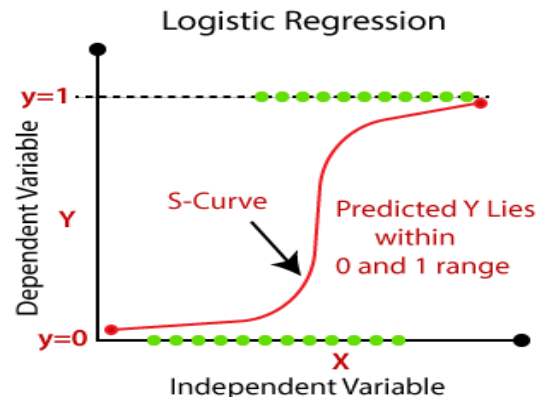
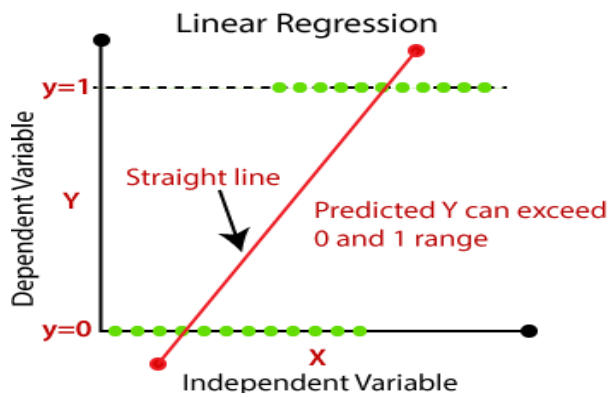
Now we will implement the Decision tree using Python. For this, we will use the dataset "**user_data.csv**," which we have used in previous classification models. By using the same dataset, we can compare the Decision tree classifier with other classification models such as [KNN](#), [SVM](#), [LogisticRegression](#),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

Linear Regression vs Logistic Regression

Linear Regression and Logistic Regression are the two famous Machine Learning Algorithms which come under supervised learning technique. Since both the algorithms are of supervised in nature hence these algorithms use labeled dataset to make the predictions. But the main difference between them is how they are being used. The Linear Regression is used for solving Regression problems whereas Logistic Regression is used for solving the Classification problems. The description of both the algorithms is given below along with difference table.



Linear Regression	Logistic Regression
Linear regression is used to predict the continuous dependent variable using a given set of independent variables.	Logistic Regression is used to predict the categorical dependent variable using a given set of independent variables.
Linear Regression is used for solving Regression problem.	Logistic regression is used for solving Classification problems.
In Linear regression, we predict the value of continuous variables.	In logistic Regression, we predict the values of categorical variables.
In linear regression, we find the best fit line, by which we can easily predict the output.	In Logistic Regression, we find the S-curve by which we can classify the samples.
Least square estimation method is used for estimation of accuracy.	Maximum likelihood estimation method is used for estimation of accuracy.
The output for Linear Regression must be a continuous value, such as price, age, etc.	The output of Logistic Regression must be a Categorical value such as 0 or 1, Yes or No, etc.
In Linear regression, it is required that relationship between dependent variable and independent variable must be linear.	In Logistic regression, it is not required to have the linear relationship between the dependent and independent variable.
In linear regression, there may be collinearity between the independent variables.	In logistic regression, there should not be collinearity between the independent variable.

UNIT-04

K-Means Clustering Algorithm

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering.

What is K-Means Algorithm?

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if $K=2$, there will be two clusters, and for $K=3$, there will be three clusters, and so on.

It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.

It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

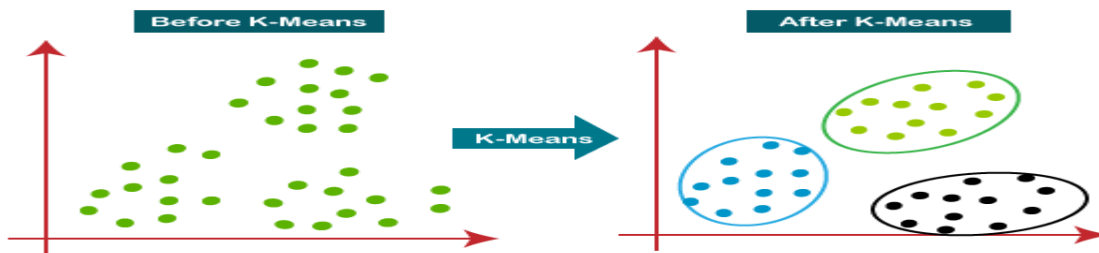
The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

The below diagram explains the working of the K-means Clustering Algorithm:



How does the K-Means Algorithm Work?

The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

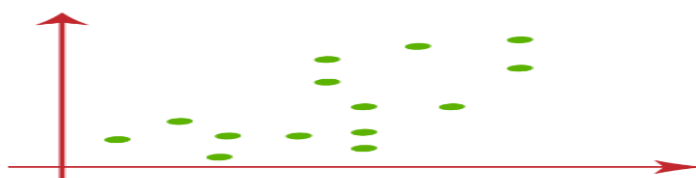
Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

Let's understand the above steps by considering the visual plots:

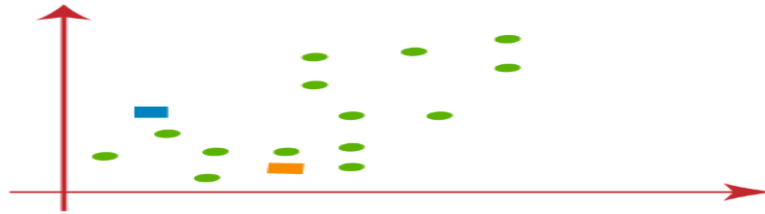
Suppose we have two variables $M1$ and $M2$. The x-y axis scatter plot of these two variables is given below:



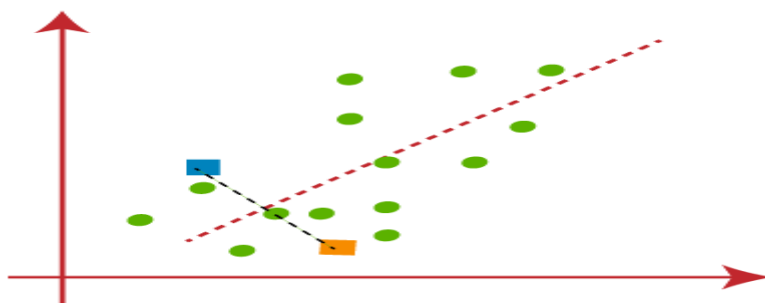
- Let's take number k of clusters, i.e., $K=2$, to identify the dataset and to put them into different clusters. It means here we will try to group these datasets into two different clusters.
- We need to choose some random k points or centroid to form the cluster. These points can be either the points from the dataset or any other point. So, here we are selecting the below two points as k points, which are not the part of our dataset. Consider the

below

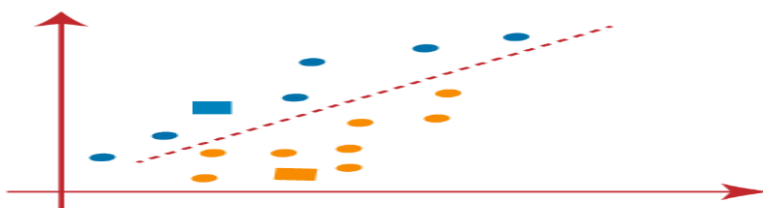
image:



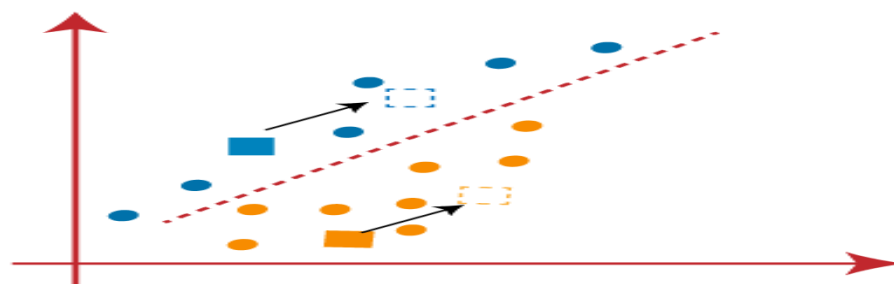
- Now we will assign each data point of the scatter plot to its closest K-point or centroid. We will compute it by applying some mathematics that we have studied to calculate the distance between two points. So, we will draw a median between both the centroids. Consider the below image:



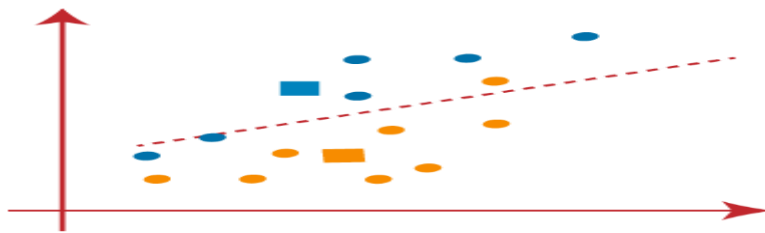
From the above image, it is clear that points left side of the line is near to the K1 or blue centroid, and points to the right of the line are close to the yellow centroid. Let's color them as blue and yellow for clear visualization.



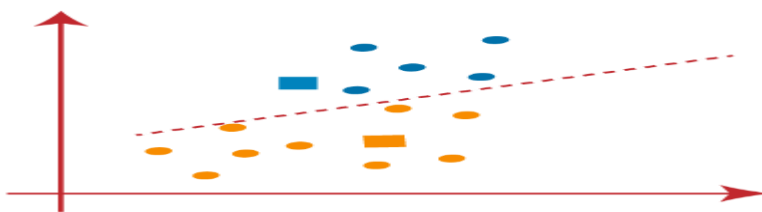
- As we need to find the closest cluster, so we will repeat the process by choosing a **new centroid**. To choose the new centroids, we will compute the center of gravity of these centroids, and will find new centroids as below:



- Next, we will reassign each datapoint to the new centroid. For this, we will repeat the same process of finding a median line. The median will be like below image:

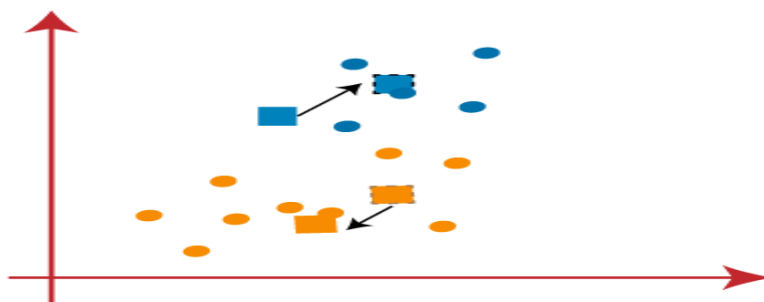


From the above image, we can see, one yellow point is on the left side of the line, and two blue points are right to the line. So, these three points will be assigned to new centroids.

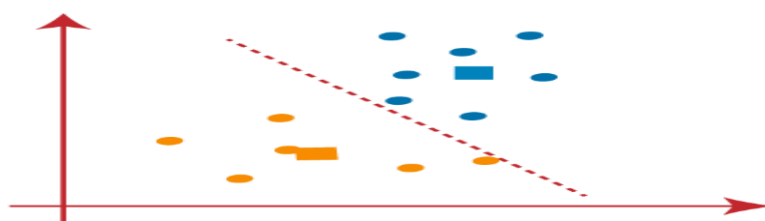


As reassignment has taken place, so we will again go to the step-4, which is finding new centroids or K-points.

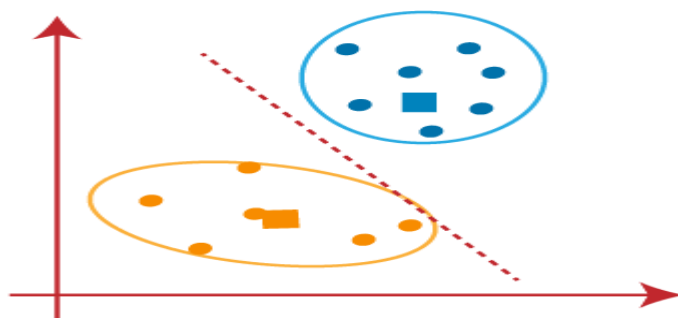
- We will repeat the process by finding the center of gravity of centroids, so the new centroids will be as shown in the below image:



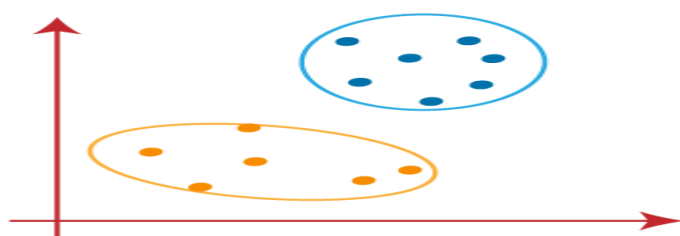
- As we got the new centroids so again will draw the median line and reassign the data points. So, the image will be:



- We can see in the above image; there are no dissimilar data points on either side of the line, which means our model is formed. Consider the below image:



As our model is ready, so we can now remove the assumed centroids, and the two final clusters will be as shown in the below image:



How to choose the value of "K number of clusters" in K-means Clustering?

The performance of the K-means clustering algorithm depends upon highly efficient clusters that it forms. But choosing the optimal number of clusters is a big task. There are some different ways to find the optimal number of clusters, but here we are discussing the most appropriate method to find the number of clusters or value of K. The method is given below:

Elbow Method

The Elbow method is one of the most popular ways to find the optimal number of clusters. This method uses the concept of WCSS value. **WCSS** stands for **Within Cluster Sum of Squares**, which defines the total variations within a cluster. The formula to calculate the value of WCSS (for 3 clusters) is given below:

$$WCSS = \sum_{P_i \text{ in Cluster1}} \text{distance}(P_i C_1)^2 + \sum_{P_i \text{ in Cluster2}} \text{distance}(P_i C_2)^2 + \sum_{P_i \text{ in Cluster3}} \text{distance}(P_i C_3)^2$$

In the above formula of WCSS,

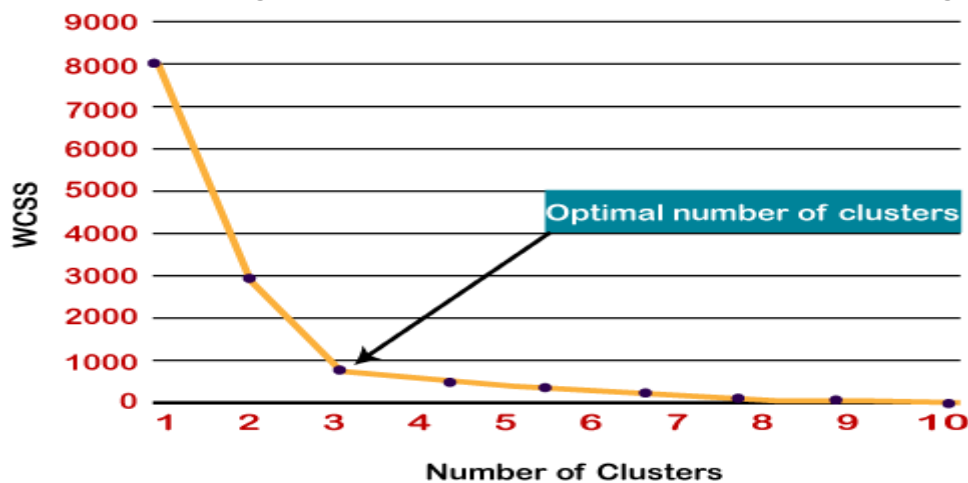
$\sum_{P_i \text{ in Cluster1}} \text{distance}(P_i C_1)^2$: It is the sum of the square of the distances between each data point and its centroid within a cluster1 and the same for the other two terms.

To measure the distance between data points and centroid, we can use any method such as Euclidean distance or Manhattan distance.

To find the optimal value of clusters, the elbow method follows the below steps:

- It executes the K-means clustering on a given dataset for different K values (ranges from 1-10).
- For each value of K, calculates the WCSS value.
- Plots a curve between calculated WCSS values and the number of clusters K.
- The sharp point of bend or a point of the plot looks like an arm, then that point is considered as the best value of K.

Since the graph shows the sharp bend, which looks like an elbow, hence it is known as the elbow method. The graph for the elbow method looks like the below image:



Note: We can choose the number of clusters equal to the given data points. If we choose the number of clusters equal to the data points, then the value of WCSS becomes zero, and that will be the endpoint of the plot.

Python Implementation of K-means Clustering Algorithm

In the above section, we have discussed the K-means algorithm, now let's see how it can be implemented using [Python](#).

Before implementation, let's understand what type of problem we will solve here. So, we have a dataset of **Mall_Customers**, which is the data of customers who visit the mall and spend there.

In the given dataset, we have **Customer_Id**, **Gender**, **Age**, **Annual Income (\$)**, and **Spending Score** (which is the calculated value of how much a customer has spent in the mall, the more the value, the more he has spent). From this dataset, we need to calculate some patterns, as it is an unsupervised method, so we don't know what to calculate exactly.

The steps to be followed for the implementation are given below:

- **Data Pre-processing**
- **Finding the optimal number of clusters using the elbow method**
- **Training the K-means algorithm on the training dataset**
- **Visualizing the clusters**

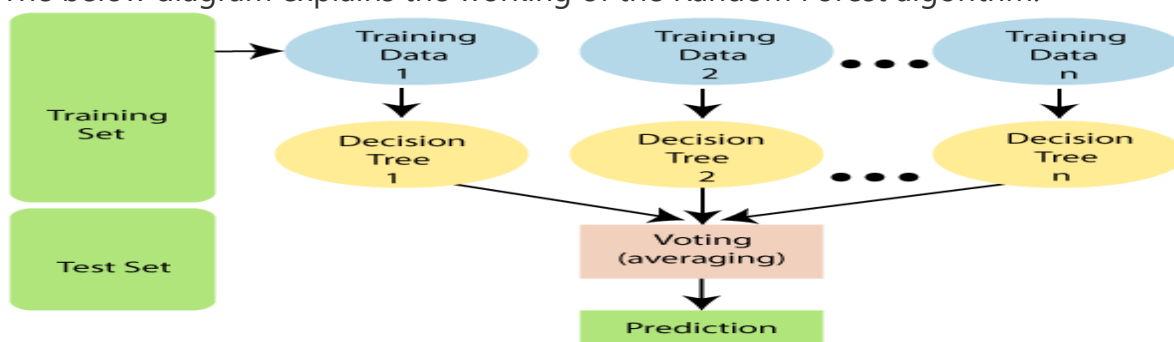
Random Forest Algorithm

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:



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).

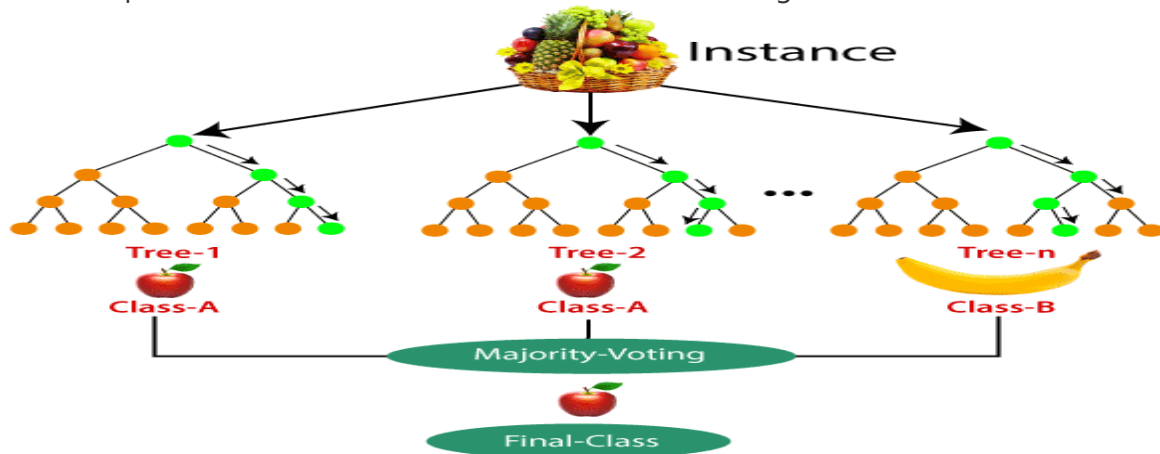
Step-3: Choose the number N for decision trees that you want to build.

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.

Confusion Matrix in Machine Learning

The confusion matrix is a matrix used to determine the performance of the classification models for a given set of test data. It can only be determined if the true values for test data are known. The matrix itself can be easily understood, but the related terminologies may be confusing. Since it shows the errors in the model performance in the form of a matrix, hence also known as an **error matrix**. Some features of Confusion matrix are given below:

- For the 2 prediction classes of classifiers, the matrix is of 2*2 table, for 3 classes, it is 3*3 table, and so on.
- The matrix is divided into two dimensions, that are **predicted values** and **actual values** along with the total number of predictions.
- Predicted values are those values, which are predicted by the model, and actual values are the true values for the given observations.
- It looks like the below table:

n = total predictions	Actual: No	Actual: Yes
Predicted: No	True Negative	False Positive
Predicted: Yes	False Negative	True Positive

The above table has the following cases:

- **True Negative:** Model has given prediction No, and the real or actual value was also No.
- **True Positive:** The model has predicted yes, and the actual value was also true.
- **False Negative:** The model has predicted no, but the actual value was Yes, it is also called as **Type-II error**.
- **False Positive:** The model has predicted Yes, but the actual value was No. It is also called a **Type-I error**.

Need for Confusion Matrix in Machine learning

- It evaluates the performance of the classification models, when they make predictions on test data, and tells how good our classification model is.

- It not only tells the error made by the classifiers but also the type of errors such as it is either type-I or type-II error.
- With the help of the confusion matrix, we can calculate the different parameters for the model, such as accuracy, precision, etc.

Example: We can understand the confusion matrix using an example.

Suppose we are trying to create a model that can predict the result for the disease that is either a person has that disease or not. So, the confusion matrix for this is given as:

n = 100	Actual: No	Actual: Yes	
Predicted: No	TN: 65	FP: 3	68
Predicted: Yes	FN: 8	TP: 24	32
	73	27	

From the above example, we can conclude that:

- The table is given for the two-class classifier, which has two predictions "Yes" and "NO." Here, Yes defines that patient has the disease, and No defines that patient does not has that disease.
- The classifier has made a total of **100 predictions**. Out of 100 predictions, **89 are true predictions**, and **11 are incorrect predictions**.
- The model has given prediction "yes" for 32 times, and "No" for 68 times. Whereas the actual "Yes" was 27, and actual "No" was 73 times.

Calculations using Confusion Matrix:

We can perform various calculations for the model, such as the model's accuracy, using this matrix. These calculations are given below:

- **Classification Accuracy:** It is one of the important parameters to determine the accuracy of the classification problems. It defines how often the model predicts the correct output. It can be calculated as the ratio of the number of correct predictions made by the classifier to all number of predictions made by the classifiers. The formula

is given below:

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+FN+TN}$$

- **Misclassification rate:** It is also termed as Error rate, and it defines how often the model gives the wrong predictions. The value of error rate can be calculated as the number of incorrect predictions to all number of the predictions made by the classifier.

The formula is given below:

$$\text{Error rate} = \frac{FP+FN}{TP+FP+FN+TN}$$

- **Precision:** It can be defined as the number of correct outputs provided by the model or out of all positive classes that have predicted correctly by the model, how many of them were actually true. It can be calculated using the below formula:

$$\text{Precision} = \frac{TP}{TP+FP}$$

- **Recall:** It is defined as the out of total positive classes, how our model predicted correctly. The recall must be as high as possible.

$$\text{Recall} = \frac{TP}{TP+FN}$$

- **F-measure:** If two models have low precision and high recall or vice versa, it is difficult to compare these models. So, for this purpose, we can use F-score. This score helps us to evaluate the recall and precision at the same time. The F-score is maximum if the recall is equal to the precision. It can be calculated using the below formula:

$$\text{F-measure} = \frac{2 * \text{Recall} * \text{Precision}}{\text{Recall} + \text{Precision}}$$

Other important terms used in Confusion Matrix:

- **Null Error rate:** It defines how often our model would be incorrect if it always predicted the majority class. As per the accuracy paradox, it is said that "*the best classifier has a higher error rate than the null error rate.*"
- **ROC Curve:** The ROC is a graph displaying a classifier's performance for all possible thresholds. The graph is plotted between the true positive rate (on the Y-axis) and the false Positive rate (on the x-axis).

AUC-ROC Curve in Machine Learning

In Machine Learning, only developing an ML model is not sufficient as we also need to see whether it is performing well or not. It means that after building an ML model, we need to evaluate and validate how good or bad it is, and for such cases, we use different Evaluation Metrics. *AUC-ROC curve is such an evaluation metric that is used to visualize the performance of a classification model.* It is one of the popular and important metrics for evaluating the performance of the classification model. In this topic, we are going to discuss more details about the AUC-ROC curve.

Note: For a better understanding of this article, we suggest you first understand [the Confusion Matrix](#), as AUC-ROC uses terminologies used in the Confusion matrix.

What is AUC-ROC Curve?

AUC-ROC curve is a performance measurement metric of a classification model at different threshold values. Firstly, let's understand ROC (Receiver Operating Characteristic curve) curve.

ROC Curve

ROC or Receiver Operating Characteristic curve represents a probability graph to show the performance of a classification model at different threshold levels. The curve is plotted between two parameters, which are:

- **True Positive Rate or TPR**
- **False Positive Rate or FPR**

In the curve, TPR is plotted on Y-axis, whereas FPR is on the X-axis.

TPR:

TPR or True Positive rate is a synonym for Recall, which can be calculated as:

$$TPR = \frac{TP}{TP + FN}$$

FPR or False Positive Rate can be calculated as:

$$FPR = \frac{FP}{FP + TN}$$

Here, **TP**: True Positive

FP: False Positive

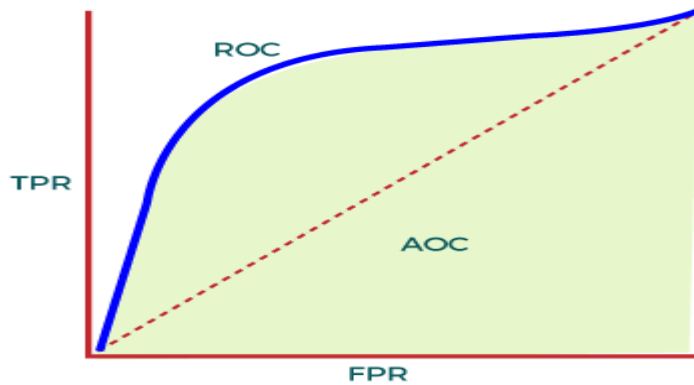
TN: True Negative

FN: False Negative

Now, to efficiently calculate the values at any threshold level, we need a method, which is AUC.

AUC: Area Under the ROC curve

AUC is known for **Area Under the ROC curve**. As its name suggests, AUC calculates the two-dimensional area under the entire ROC curve ranging from (0,0) to (1,1), as shown below image:



In the ROC curve, AUC computes the performance of the binary classifier across different thresholds and provides an aggregate measure. The value of AUC ranges from 0 to 1, which means an excellent model will have AUC near 1, and hence it will show a good measure of Separability.

When to Use AUC-ROC

AUC is preferred due to the following cases:

- AUC is used to measure how well the predictions are ranked instead of giving their absolute values. Hence, we can say AUC is **Scale-Invariant**.
- It measures the quality of predictions of the model without considering the selected classification threshold. It means AUC is **classification-threshold-invariant**.

When not to use AUC-ROC

- AUC is not preferable when we need to calibrate probability output.
- Further, AUC is not a useful metric when there are wide disparities in the cost of false negatives vs false positives, and it is difficult to minimize one type of classification error.

How AUC-ROC curve can be used for the Multi-class Model?

Although the AUC-ROC curve is only used for binary classification problems, we can also use it for multiclass classification problems. For multi-class classification problems, we can plot N number of AUC curves for N number of classes with the One vs ALL method.

For example, if we have three different classes, X, Y, and Z, then we can plot a curve for X against Y & Z, a second plot for Y against X & Z, and the third plot for Z against Y and X.

Applications of AUC-ROC Curve

Although the AUC-ROC curve is used to evaluate a classification model, it is widely used for various applications. Some of the important applications of AUC-ROC are given below:

1. **Classification of 3D model**
The curve is used to classify a 3D model and separate it from the normal models. With the specified threshold level, the curve classifies the non-3D and separates out the 3D models.
2. **Healthcare**
The curve has various applications in the healthcare sector. It can be used to detect cancer disease in patients. It does this by using false positive and false negative rates, and accuracy depends on the threshold value used for the curve.
3. **Binary Classification**
AUC-ROC curve is mainly used for binary classification problems to evaluate their performance

Mean Absolute Deviation (MAD)

The mean absolute deviation (MAD) of a data-set is the average distance between each data point of the data-set and the mean of data. i.e it represents the amount of variation that occurs around the mean value in the data-set. It is also a measure of variation. It is calculated as the average of the sum of the absolute difference between each value of the data-set and the mean.

$$\text{MAD} = (\sum |x_i - \text{mean}|) \div n$$

where $1 < i < n$ and n is the number of data-points in the data-set

Example 1: The data-set is 11 , 15 , 18 , 17 , 12 , 17. Calculate the mean absolute deviation of the given data-set?

Solution:

Step 1: Calculating the mean

$$\bar{x} = (x_1 + x_2 + x_3 + \dots + x_n) / n$$

$$\bar{x} = (11 + 15 + 18 + 17 + 12 + 17) / 6$$

$$\bar{x} = 15$$

The mean of the given data = 15

Step 2: Calculating the absolute difference between each data-point and mean.

Data-Point	Absolute Difference from mean
11	$ 11 - 15 = 4$
12	$ 12 - 15 = 3$
15	$ 15 - 15 = 0$
17	$ 17 - 15 = 2$
17	$ 17 - 15 = 2$
18	$ 18 - 15 = 3$

Step 3: Adding the Absolute Difference together

$$(\sum |x_i - \text{mean}|) = 4 + 3 + 0 + 2 + 2 + 3$$

$$(\sum |x_i - \text{mean}|) = 14$$

Step 4: Dividing the sum of absolute difference and the number of data-points.

$$\text{MAD} = (\sum |x_i - \text{mean}|) \div n$$

$$\text{MAD} = 14/6$$

$$\text{MAD} = 2.33$$

Hence, we can conclude that, on average, each data-point is 2 distance away from the mean.

Example 2: The following table shows the number of oranges that grew on Nancy's orange tree each season

Season	Number of Oranges
Winter	5
Summer	17
Spring	24
Fall	10

Find the mean absolute deviation (MAD) of the data set?

Solution:

Step 1: Calculating the mean

$$\bar{x} = (x_1 + x_2 + x_3 + \dots + x_n) / n$$

$$\bar{x} = (5 + 17 + 24 + 10) / 4$$

$$\bar{x} = 56/4$$

The mean of the given data = 14

Step 2: Calculating the absolute difference between each data-point and mean

Data-Point	Absolute Difference from mean
5	$ 5 - 14 = 9$
17	$ 17 - 14 = 3$
24	$ 24 - 14 = 10$
10	$ 10 - 14 = 4$

Step 3: Adding the Absolute Difference together

$$(\sum |x_i - \text{mean}|) = 9 + 3 + 10 + 4$$

$$(\sum |x_i - \text{mean}|) = 26$$

Step 4: Dividing the sum of absolute difference and the number of data-points

$$MAD = (\sum |x_i - \text{mean}|) \div n$$

$$MAD = 26 / 4$$

$$MAD = 6.5$$

Example 3: Consider the following data-set

Name of the student	Marks in Maths
Chetan	90
Shubham	74

Name of the student	Marks in Maths
Riya	80
Manu	92

Calculate the mean absolute deviation of the given data?

Solution:

Step 1: Calculating the mean

$$\bar{x} = (x_1 + x_2 + x_3 + \dots + x_n) / n$$

$$\bar{x} = (90 + 74 + 80 + 92) / 4$$

$$\bar{x} = 336/4$$

The mean of the given data = 84

Step 2: Calculating the absolute difference between each data-point and mean

Data-Point	Absolute Difference from mean
90	$ 90 - 84 = 6$
74	$ 74 - 84 = 10$
80	$ 80 - 84 = 4$
92	$ 92 - 84 = 8$

Step 3: Adding the Absolute Difference together

$$(\sum |x_i - \text{mean}|) = 6 + 10 + 4 + 8$$

$$(\sum |x_i - \text{mean}|) = 28$$

Step 4: Dividing the sum of absolute difference and the number of data-points

$$MAD = (\sum |x_i - \text{mean}|) \div n$$

$$MAD = 28 / 4$$

$$MAD = 7$$