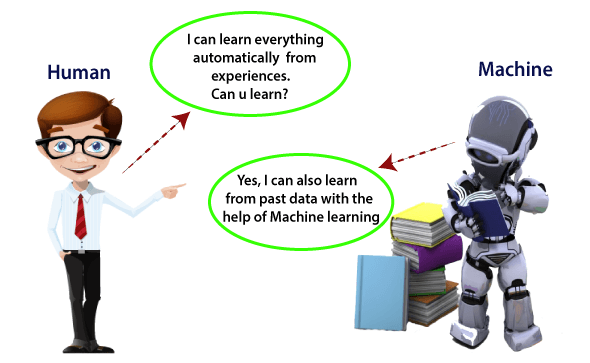
What is ML?

Machine learning is a growing technology which enables computers to learn automatically from past data. Machine learning uses various algorithms for **building mathematical models and making predictions using historical data or information**. Currently, it is being used for various tasks such as **image recognition**, **speech recognition**, **email filtering**, **Facebook auto-tagging**, **recommender system**, and many more.

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**.



How ML work with AI?

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.

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:

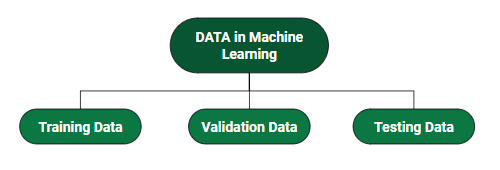
* Machine learning uses data to detect various patterns in a given dataset.
* It can learn from past data and improve automatically.
* It is a data-driven technology.
* Machine learning is much similar to data mining as it also deals with the huge amount of the data.

# ML | Introduction to Data in Machine Learning

**DATA :** It can be any unprocessed fact, value, text, sound or picture that is not being interpreted and analyzed. Data is the most important part of all Data Analytics, Machine Learning, Artificial Intelligence. Without data, we can’t train any model and all modern research and automation will go vain. Big Enterprises are spending lots of money just to gather as much certain data as possible.  
**Example:** Why did Facebook acquire WhatsApp by paying a huge price of $19 billion?  
The answer is very simple and logical – it is to have access to the users’ information that Facebook may not have but WhatsApp will have. This information of their users is of paramount importance to Facebook as it will facilitate the task of improvement in their services.  
**INFORMATION :** Data that has been interpreted and manipulated and has now some meaningful inference for the users.  
**KNOWLEDGE :** Combination of inferred information, experiences, learning and insights. Results in awareness or concept building for an individual or organization.

**How we split data in Machine Learning?**

* **Training Data:**The part of data we use to train our model. This is the data which your model actually sees(both input and output) and learn from.
* **Validation Data:**The part of data which is used to do a frequent evaluation of model, fit on training dataset along with improving involved hyperparameters (initially set parameters before the model begins learning). This data plays it’s part when the model is actually training.
* **Testing Data:**Once our model is completely trained, testing data provides the unbiased evaluation. When we feed in the inputs of Testing data, our model will predict some values(without seeing actual output). After prediction, we evaluate our model by comparing it with actual output present in the testing data. This is how we evaluate and see how much our model has learned from the experiences feed in as training data, set at the time of training.



# Machine Learning – Applications

Machine learning is one of the most exciting technologies that one would have ever come across. As it is evident from the name, it gives the computer that which makes it more similar to humans: The ability to learn. Machine learning is actively being used today, perhaps in many more places than one would expect. We probably use a learning algorithm dozens of time without even knowing it. Applications of Machine Learning include:

* **Web Search Engine:** One of the reasons why search engines like google, bing etc work so well is because the system has learnt how to rank pages through a complex learning algorithm.
* **Photo tagging Applications:** Be it facebook or any other photo tagging application, the ability to tag friends makes it even more happening. It is all possible because of a face recognition algorithm that runs behind the application.
* **Spam Detector:** Our mail agent like Gmail or Hotmail does a lot of hard work for us in classifying the mails and moving the spam mails to spam folder. This is again achieved by a spam classifier running in the back end of mail application.
* 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**.

### 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.

### 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.

### 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.

### 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.

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 classifieds into two categories of algorithms:

* **Clustering**
* **Association**

### 3) Reinforcement Learning

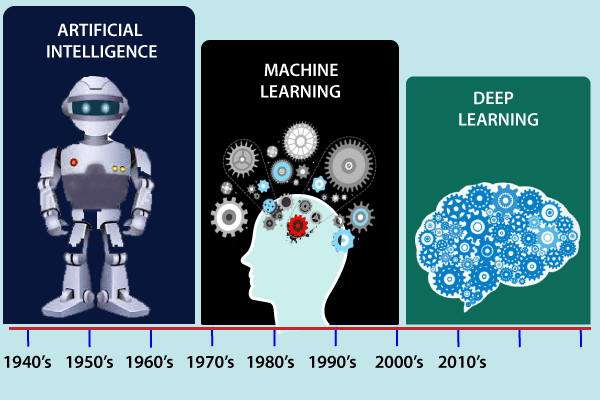
Reinforcement learning is a feedback-based learning method, in which a learning agent gets a reward for each right action and gets a penalty for each wrong action. The agent learns automatically with these feedbacks and improves its performance. In reinforcement learning, the agent interacts with the 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.

#### Note: We will learn about the above types of machine learning in detail in later chapters.

## 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:



Best Python Library for ML

Machine Learning, as the name suggests, is the science of programming a computer by which they are able to learn from different kinds of data. A more general definition given by Arthur Samuel is – “Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed.” They are typically used to solve various types of life problems.  
In the older days, people used to perform Machine Learning tasks by manually coding all the algorithms and mathematical and statistical formula. This made the process time consuming, tedious and inefficient. But in the modern days, it is become very much easy and efficient compared to the olden days by various python libraries, frameworks, and modules. Today, Python is one of the most popular programming languages for this task and it has replaced many languages in the industry, one of the reason is its vast collection of libraries. Python libraries that used in Machine Learning are:

* Numpy
* Scipy
* Scikit-learn
* Theano
* TensorFlow
* Keras
* PyTorch
* Pandas
* Matplotlib

# 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**
* **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.

# How to get datasets for Machine Learning

The key to success in the field of machine learning or to become a great data scientist is to practice with different types of datasets. But discovering a suitable dataset for each kind of machine learning project is a difficult task. So, in this topic, we will provide the detail of the sources from where you can easily get the dataset according to your project.

Before knowing the sources of the machine learning dataset, let's discuss datasets.

## 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.

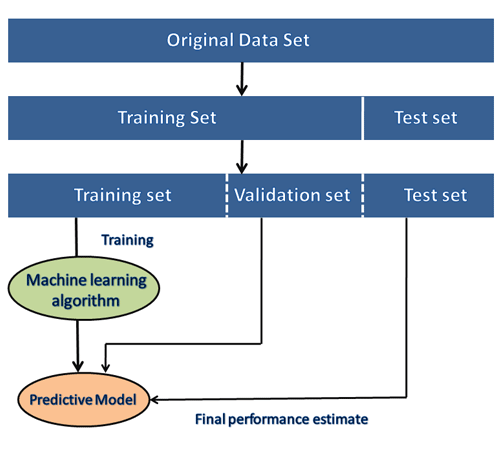
## Need of Dataset

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**

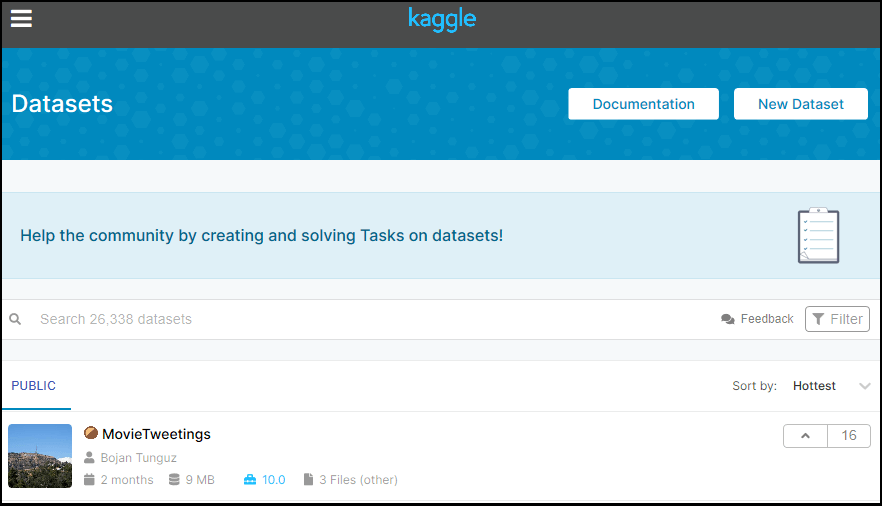


#### Note: The datasets are of large size, so to download these datasets, you must have fast internet on your computer.

## 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

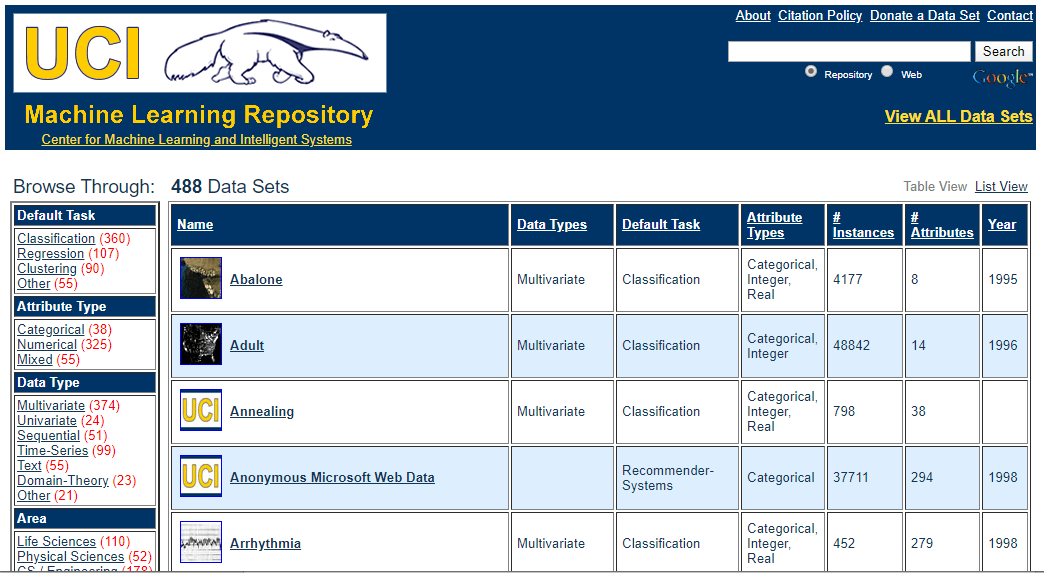


Kaggle is one of the best sources for providing datasets for Data Scientists and Machine Learners. It allows users to find, download, and publish datasets in an easy way. It also provides the opportunity to work with other machine learning engineers and solve difficult Data Science related tasks.

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



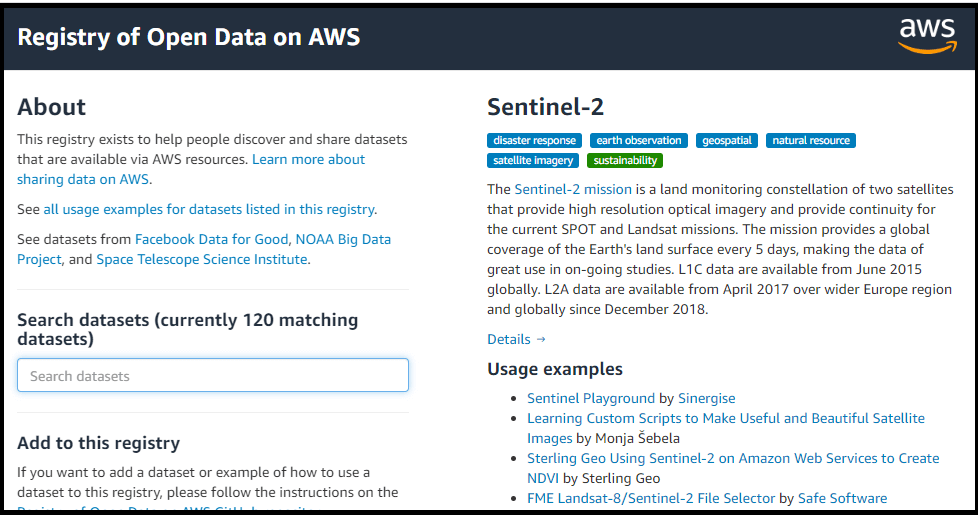
UCI Machine learning repository is one of the great sources of machine learning datasets. This repository contains databases, domain theories, and data generators that are widely used by the machine learning community for the analysis of ML algorithms.

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



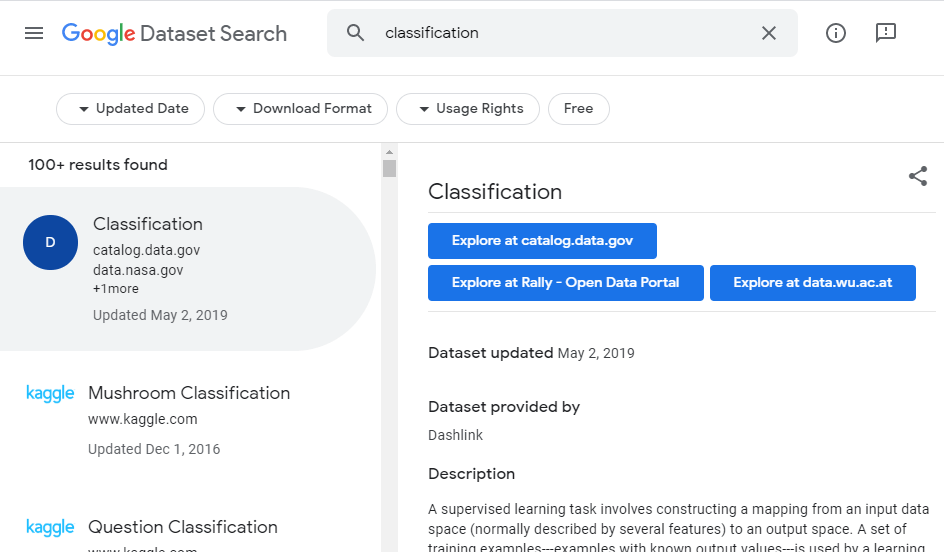
We can search, download, access, and share the datasets that are publicly available via AWS resources. These datasets can be accessed through AWS resources but provided and maintained by different government organizations, researches, businesses, or individuals.

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/>.

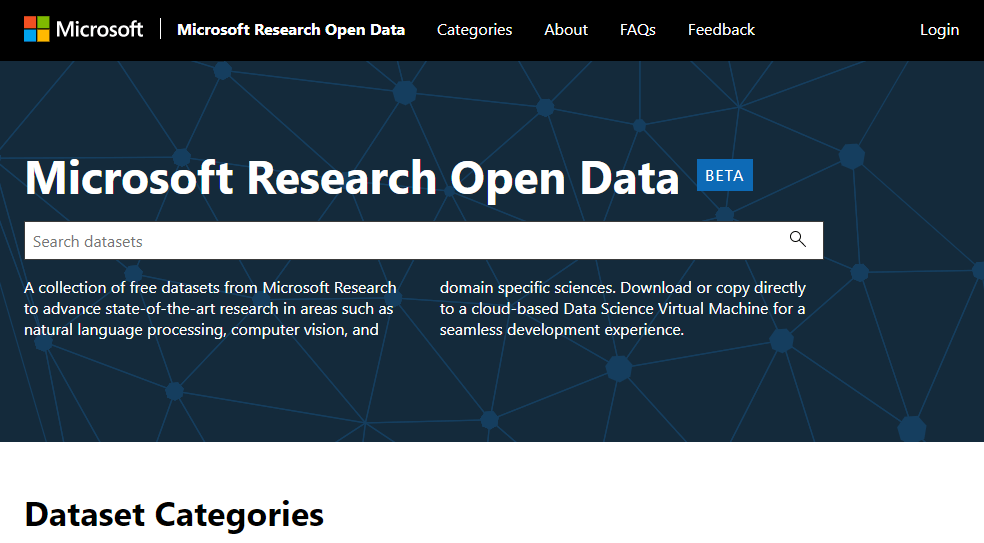
### 4. Google's Dataset Search Engine



**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>.

### 5. Microsoft Datasets



The Microsoft has launched the **"Microsoft Research Open data"** repository with the collection of free datasets in various areas such as **natural language processing, computer vision, and domain-specific sciences.**

Using this resource, we can download the datasets to use on the current device, or we can also directly use it on the cloud infrastructure.

The link to download or use the dataset from this resource is <https://msropendata.com/>.

# 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.

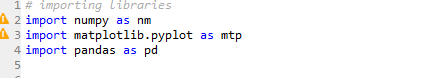
**Matplotlib:** The second library is **matplotlib**, which is a Python 2D plotting library, and with this library, we need to import a sub-library **pyplot**. This library is used to plot any type of charts in Python for the code. It will be imported as below:

1. import matplotlib.pyplot as mpt

Here we have used mpt 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:



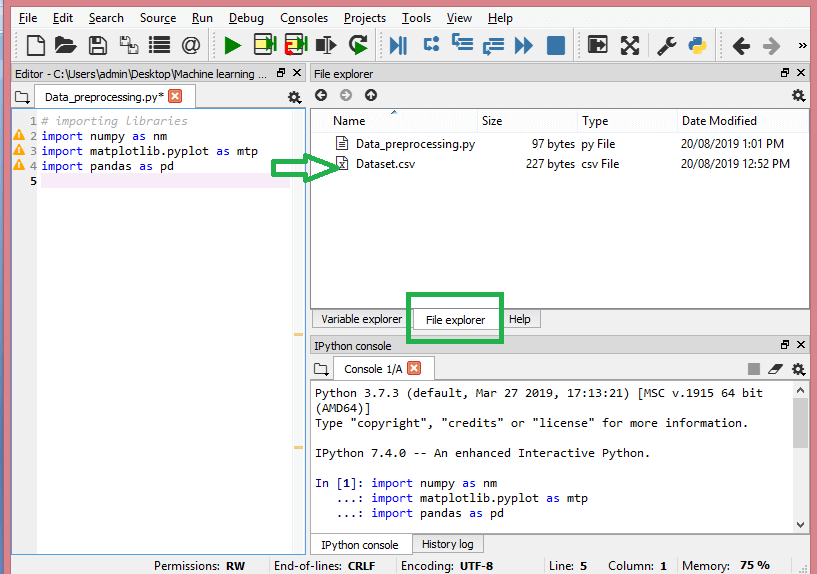
## 3) Importing the Datasets

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.



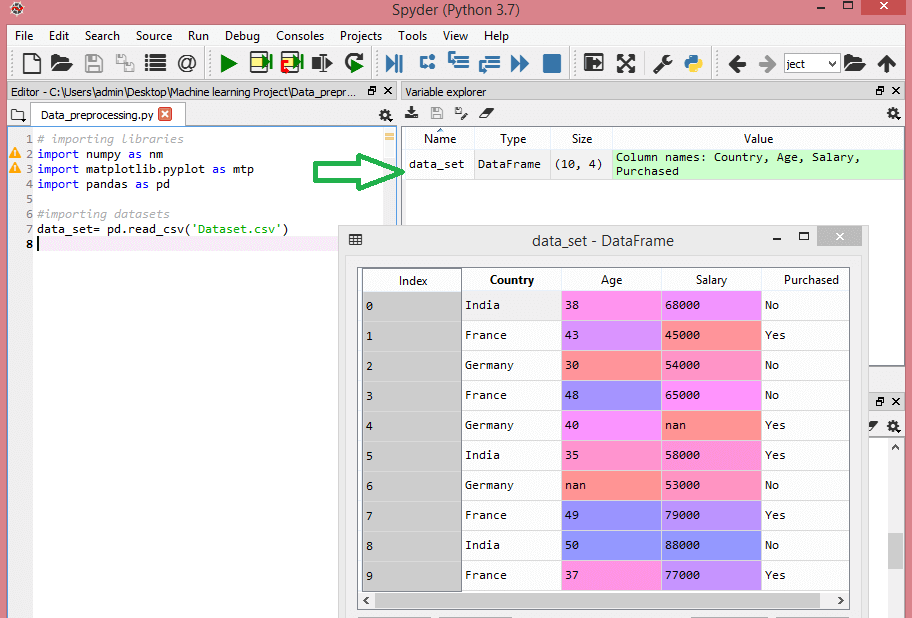
**read\_csv() function:**

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:



As in the above image, indexing is started from 0, which is the default indexing in Python. We can also change the format of our dataset by clicking on the format option.

**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. #Catgorical 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,

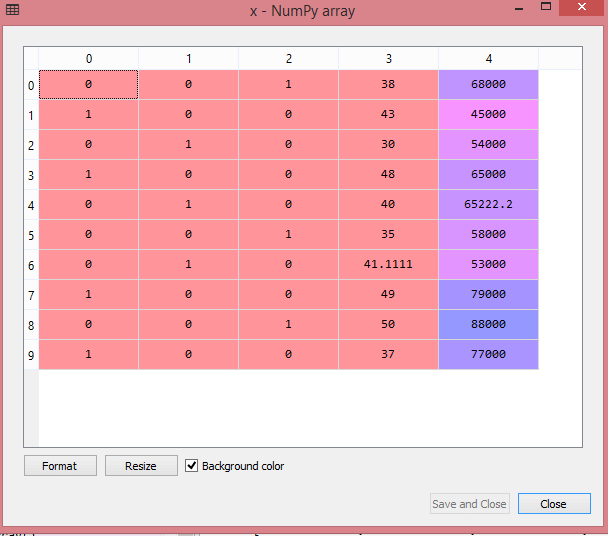
8.80000000e+04],

[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:



**For Purchased Variable:**

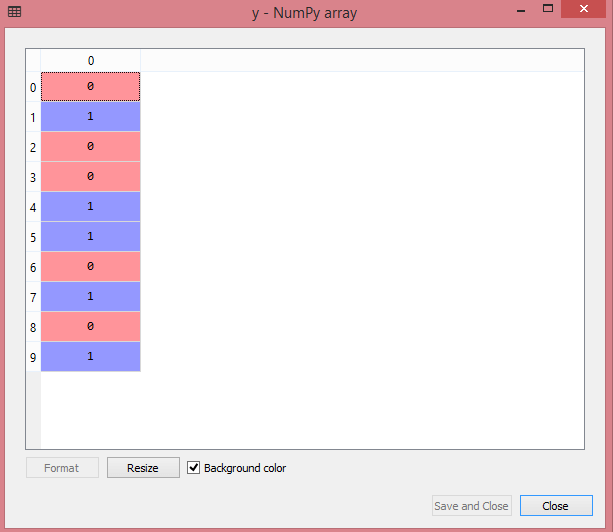
1. labelencoder\_y= LabelEncoder()
2. y= labelencoder\_y.fit\_transform(y)

For the second categorical variable, we will only use labelencoder object of **LableEncoder** 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:**

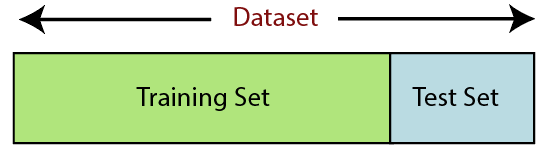


## 6) Splitting the Dataset into the Training set and Test set

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:

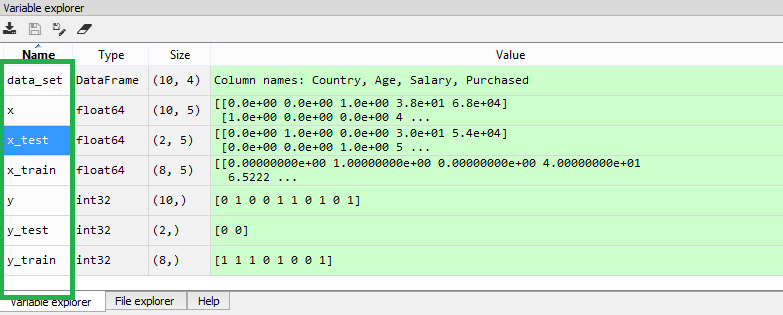
1. from sklearn.model\_selection import train\_test\_split
2. x\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size= 0.2, random\_state=0)

**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.

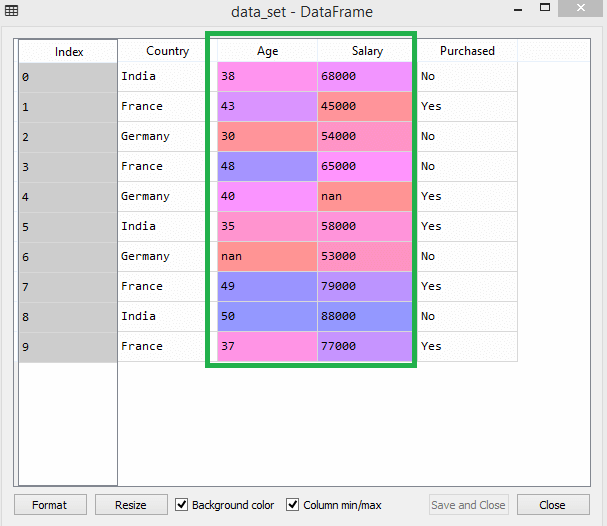


As we can see in the above image, the x and y variables are divided into 4 different variables with corresponding values.

## 7) Feature Scaling

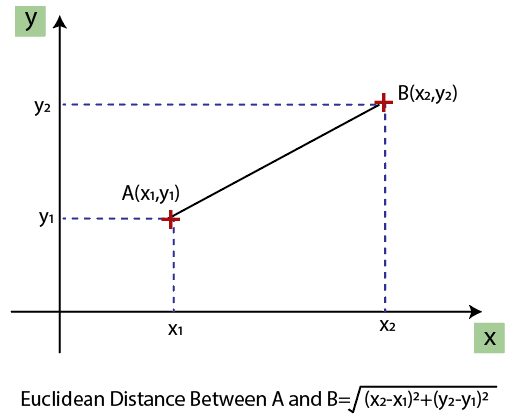
Feature scaling is the final step of data preprocessing in machine learning. It is a technique to standardize the independent variables of the dataset in a specific range. In feature scaling, we put our variables in the same range and in the same scale so that no any variable dominate the other variable.

Consider the below dataset:



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.

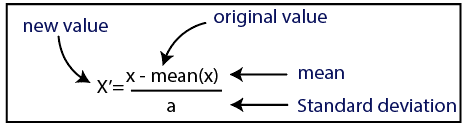
Euclidean distance is given as:



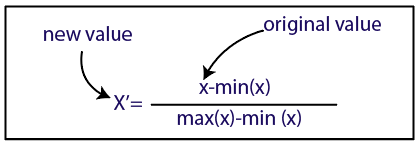
If we compute any two values from age and salary, then salary values will dominate the age values, and it will produce an incorrect result. So to remove this issue, we need to perform feature scaling for machine learning.

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

**Standardization**



**Normalization**



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)

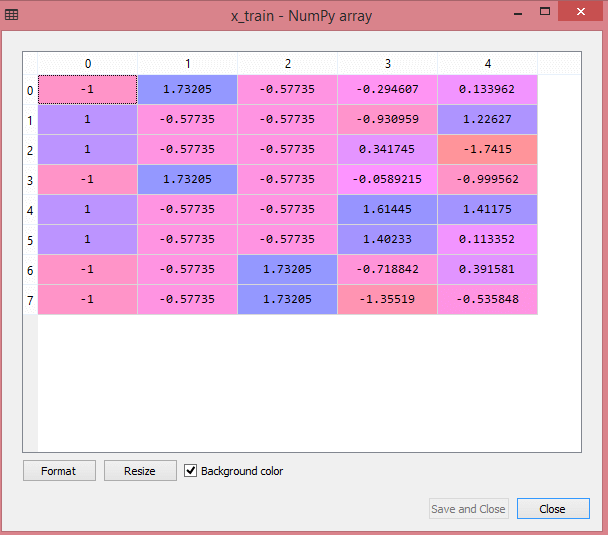
For test dataset, we will directly apply **transform()** function instead of **fit\_transform()** because it is already done in training set.

1. x\_test= st\_x.transform(x\_test)

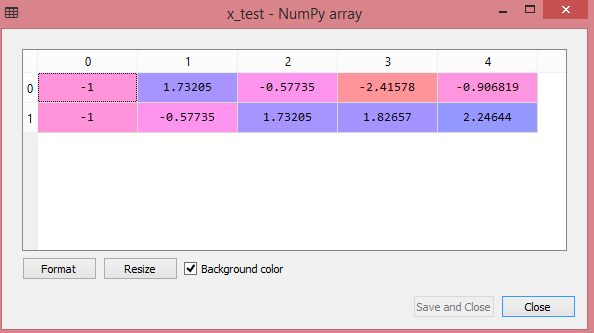
**Output:**

By executing the above lines of code, we will get the scaled values for x\_train and x\_test as:

**x\_train:**



**x\_test:**



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.

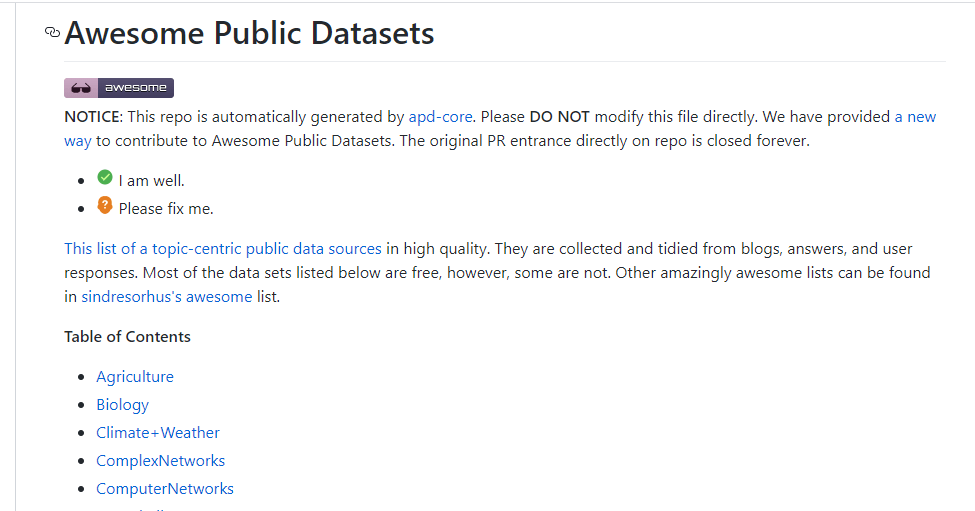
**Combining all the steps:**

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 nm
3. import matplotlib.pyplot as mtp
4. import pandas as pd
6. #importing datasets
7. data\_set= pd.read\_csv('Dataset.csv')
9. #Extracting Independent Variable
10. x= data\_set.iloc[:, :-1].values
12. #Extracting Dependent variable
13. y= data\_set.iloc[:, 3].values
15. #handling missing data(Replacing missing data with the mean value)
16. from sklearn.preprocessing import Imputer
17. imputer= Imputer(missing\_values ='NaN', strategy='mean', axis = 0)
19. #Fitting imputer object to the independent varibles x.
20. imputerimputer= imputer.fit(x[:, 1:3])
22. #Replacing missing data with the calculated mean value
23. x[:, 1:3]= imputer.transform(x[:, 1:3])
25. #for Country Variable
26. from sklearn.preprocessing import LabelEncoder, OneHotEncoder
27. label\_encoder\_x= LabelEncoder()
28. x[:, 0]= label\_encoder\_x.fit\_transform(x[:, 0])
30. #Encoding for dummy variables
31. onehot\_encoder= OneHotEncoder(categorical\_features= [0])
32. x= onehot\_encoder.fit\_transform(x).toarray()
34. #encoding for purchased variable
35. labelencoder\_y= LabelEncoder()
36. y= labelencoder\_y.fit\_transform(y)
38. # Splitting the dataset into training and test set.
39. from sklearn.model\_selection import train\_test\_split
40. x\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size= 0.2, random\_state=0)
42. #Feature Scaling of datasets
43. from sklearn.preprocessing import StandardScaler
44. st\_x= StandardScaler()
45. x\_train= st\_x.fit\_transform(x\_train)
46. x\_test= st\_x.transform(x\_test)

In the above code, we have included all the data preprocessing steps together. But there are some steps or lines of code which are not necessary for all machine learning models. So we can exclude them from our code to make it reusable for all models.

### 6. Awesome Public Dataset Collection



Awesome public dataset collection provides high-quality datasets that are arranged in a well-organized manner within a list according to topics such as Agriculture, Biology, Climate, Complex networks, etc. Most of the datasets are available free, but some may not, so it is better to check the license before downloading the dataset.

The link to download the dataset from Awesome public dataset collection is <https://github.com/awesomedata/awesome-public-datasets>.

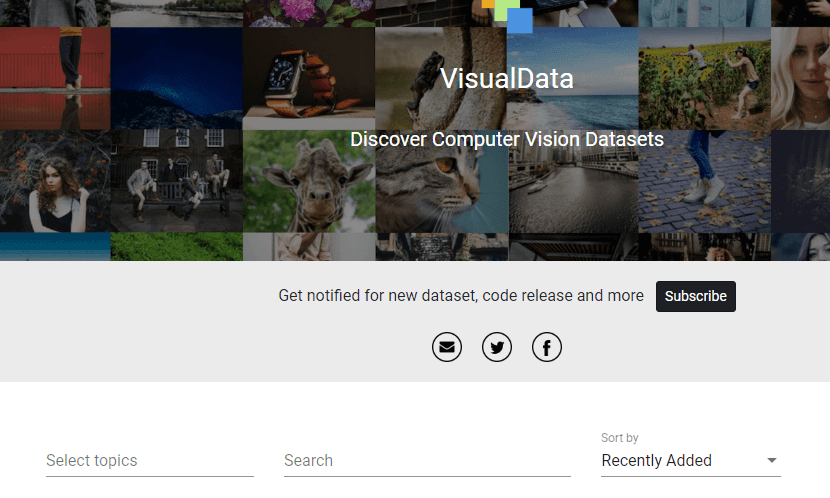
### 7. Government Datasets

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

The goal of providing these datasets is to increase transparency of government work among the people and to use the data in an innovative approach. Below are some links of government datasets:

* [Indian Government dataset](https://data.gov.in/)
* [US Government Dataset](https://www.data.gov/)
* [Northern Ireland Public Sector Datasets](https://www.opendatani.gov.uk/)
* [European Union Open Data Portal](https://data.europa.eu/euodp/data/dataset)

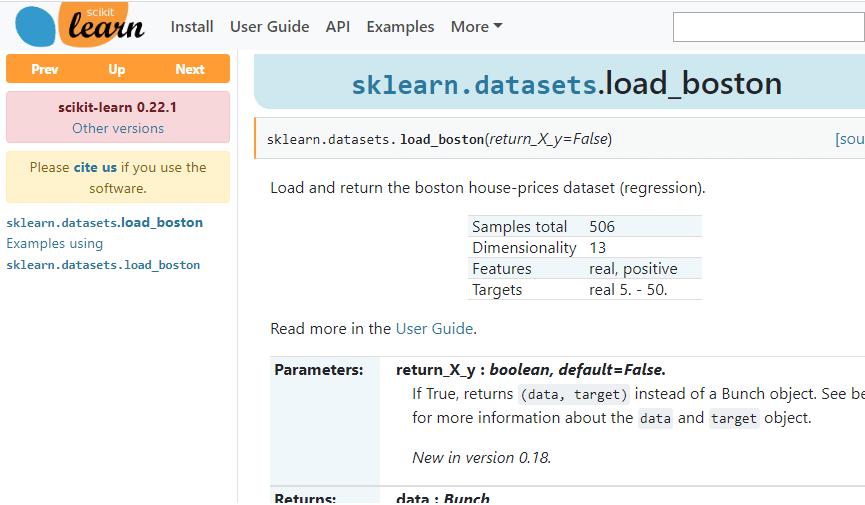
### 8. Computer Vision Datasets



Visual data provides multiple numbers of the great dataset that are specific to computer visions such as Image Classification, Video classification, Image Segmentation, etc. Therefore, if you want to build a project on deep learning or image processing, then you can refer to this source.

The link for downloading the dataset from this source is <https://www.visualdata.io/>.

### 9. Scikit-learn dataset



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.

The link to download datasets from this source is <https://scikit-learn.org/stable/datasets/index.html>.

# 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

#### Note: We will discuss these algorithms in detail in later chapters.

## 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.

# Unsupervised Machine Learning

In the previous topic, we learned supervised machine learning in which models are trained using labeled data under the supervision of training data. But there may be many cases in which we do not have labeled data and need to find the hidden patterns from the given dataset. So, to solve such types of cases in machine learning, we need unsupervised learning techniques.

## 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:

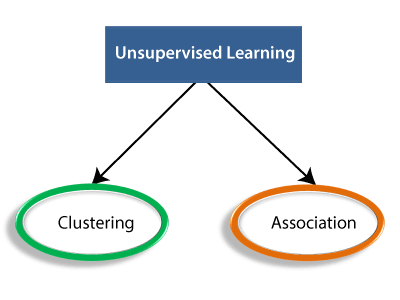


Here, we have taken an unlabeled input data, which means it is not categorized and corresponding outputs are also not given. Now, this unlabeled input data is fed to the machine learning model in order to train it. Firstly, it will interpret the raw data to find the hidden patterns from the data and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

#### Note: We will learn these algorithms in later chapters.

## Unsupervised Learning algorithms:

Below is the list of some popular unsupervised learning algorithms:

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

## Advantages of Unsupervised Learning

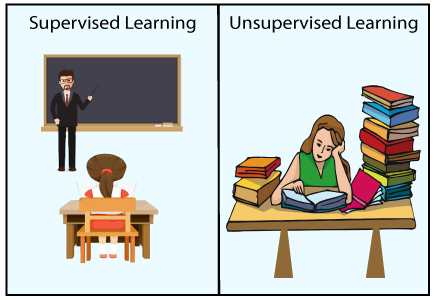
* Unsupervised learning is used for more complex tasks as compared to supervised learning because, in unsupervised learning, we don't have labeled input data.
* Unsupervised learning is preferable as it is easy to get unlabeled data in comparison to labeled data.

## 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.

# Difference between Supervised and Unsupervised Learning

Supervised and Unsupervised learning are the two techniques of machine learning. But both the techniques are used in different scenarios and with different datasets. Below the explanation of both learning methods along with their difference table is given.



## Supervised Machine Learning:

Supervised learning is a machine learning method in which models are trained using labeled data. In supervised learning, models need to find the mapping function to map the input variable (X) with the output variable (Y).

Supervised Machine learning

Supervised learning needs supervision to train the model, which is similar to as a student learns things in the presence of a teacher. Supervised learning can be used for two types of problems: **Classification** and **Regression**.

**Learn more** [Supervised Machine Learning](https://www.javatpoint.com/supervised-machine-learning)

**Example:** Suppose we have an image of different types of fruits. The task of our supervised learning model is to identify the fruits and classify them accordingly. So to identify the image in supervised learning, we will give the input data as well as output for that, which means we will train the model by the shape, size, color, and taste of each fruit. Once the training is completed, we will test the model by giving the new set of fruit. The model will identify the fruit and predict the output using a suitable algorithm.

## Unsupervised Machine Learning:

Unsupervised learning is another machine learning method in which patterns inferred from the unlabeled input data. The goal of unsupervised learning is to find the structure and patterns from the input data. Unsupervised learning does not need any supervision. Instead, it finds patterns from the data by its own.

**Learn more** [Unsupervised Machine Learning](https://www.javatpoint.com/unsupervised-machine-learning)

Unsupervised learning can be used for two types of problems: **Clustering** and **Association**.

**Example:** To understand the unsupervised learning, we will use the example given above. So unlike supervised learning, here we will not provide any supervision to the model. We will just provide the input dataset to the model and allow the model to find the patterns from the data. With the help of a suitable algorithm, the model will train itself and divide the fruits into different groups according to the most similar features between them.

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. |

# Data Loading for ML Projects

## Consideration While Loading CSV data

CSV data format is the most common format for ML data, but we need to take care about following major considerations while loading the same into our ML projects −

### File Header

In CSV data files, the header contains the information for each field. We must use the same delimiter for the header file and for data file because it is the header file that specifies how should data fields be interpreted.

The following are the two cases related to CSV file header which must be considered −

* **Case-I: When Data file is having a file header** − It will automatically assign the names to each column of data if data file is having a file header.
* **Case-II: When Data file is not having a file header** − We need to assign the names to each column of data manually if data file is not having a file header.

In both the cases, we must need to specify explicitly weather our CSV file contains header or not.

### Comments

Comments in any data file are having their significance. In CSV data file, comments are indicated by a hash (#) at the start of the line. We need to consider comments while loading CSV data into ML projects because if we are having comments in the file then we may need to indicate, depends upon the method we choose for loading, whether to expect those comments or not.

### Delimiter

In CSV data files, comma (,) character is the standard delimiter. The role of delimiter is to separate the values in the fields. It is important to consider the role of delimiter while uploading the CSV file into ML projects because we can also use a different delimiter such as a tab or white space. But in the case of using a different delimiter than standard one, we must have to specify it explicitly.

### Quotes

In CSV data files, double quotation (“ ”) mark is the default quote character. It is important to consider the role of quotes while uploading the CSV file into ML projects because we can also use other quote character than double quotation mark. But in case of using a different quote character than standard one, we must have to specify it explicitly.

## Methods to Load CSV Data File

While working with ML projects, the most crucial task is to load the data properly into it. The most common data format for ML projects is CSV and it comes in various flavors and varying difficulties to parse. In this section, we are going to discuss about three common approaches in Python to load CSV data file −

### Load CSV with Python Standard Library

The first and most used approach to load CSV data file is the use of Python standard library which provides us a variety of built-in modules namely **csv module** and the reader()function. The following is an example of loading CSV data file with the help of it −

**Example**

In this example, we are using the iris flower data set which can be downloaded into our local directory. After loading the data file, we can convert it into **NumPy** array and use it for ML projects. Following is the Python script for loading CSV data file −

First, we need to import the csv module provided by Python standard library as follows −

import csv

Next, we need to import Numpy module for converting the loaded data into NumPy array.

import numpy as np

Now, provide the full path of the file, stored on our local directory, having the CSV data file −

path = r"c:\iris.csv"

Next, use the csv.reader()function to read data from CSV file −

with open(path,'r') as f:

reader = csv.reader(f,delimiter = ',')

headers = next(reader)

data = list(reader)

data = np.array(data).astype(float)

We can print the names of the headers with the following line of script −

print(headers)

The following line of script will print the shape of the data i.e. number of rows & columns in the file −

print(data.shape)

Next script line will give the first three line of data file −

print(data[:3])

**Output**

['sepal\_length', 'sepal\_width', 'petal\_length', 'petal\_width']

(150, 4)

[ [5.1 3.5 1.4 0.2]

[4.9 3. 1.4 0.2]

[4.7 3.2 1.3 0.2]

]

## Load CSV with NumPy

Another approach to load CSV data file is NumPy and numpy.loadtxt() function. The following is an example of loading CSV data file with the help of it −

### Example

In this example, we are using the Pima Indians Dataset having the data of diabetic patients. This dataset is a numeric dataset with no header. It can also be downloaded into our local directory. After loading the data file, we can convert it into NumPy array and use it for ML projects. The following is the Python script for loading CSV data file −

from numpy import loadtxt

path = r"C:\pima-indians-diabetes.csv"

datapath= open(path, 'r')

data = loadtxt(datapath, delimiter=",")

print(data.shape)

print(data[:3])

### Output

(768, 9)

[ [ 6. 148. 72. 35. 0. 33.6 0.627 50. 1.]

[ 1. 85. 66. 29. 0. 26.6 0.351 31. 0.]

[ 8. 183. 64. 0. 0. 23.3 0.672 32. 1.]

]

## Load CSV with Pandas

Another approach to load CSV data file is by **Pandas** and **pandas.read\_csv()function**. This is the very flexible function that returns a **pandas.DataFrame** which can be used immediately for plotting. The following is an example of loading CSV data file with the help of it −

### Example

Here, we will be implementing two Python scripts, first is with Iris data set having headers and another is by using the Pima Indians Dataset which is a numeric dataset with no header. Both the datasets can be downloaded into local directory.

**Script-1**

The following is the Python script for loading CSV data file using Pandas on Iris Data set −

from pandas import read\_csv

path = r"C:\iris.csv"

data = read\_csv(path)

print(data.shape)

print(data[:3])

Output:

(150, 4)

sepal\_length sepal\_width petal\_length petal\_width

0 5.1 3.5 1.4 0.2

1 4.9 3.0 1.4 0.2

2 4.7 3.2 1.3 0.2

**Script-2**

The following is the Python script for loading CSV data file, along with providing the headers names too, using Pandas on Pima Indians Diabetes dataset −

from pandas import read\_csv

path = r"C:\pima-indians-diabetes.csv"

headernames = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

data = read\_csv(path, names=headernames)

print(data.shape)

print(data[:3])

**Output**

(768, 9)

preg plas pres skin test mass pedi age class

0 6 148 72 35 0 33.6 0.627 50 1

1 1 85 66 29 0 26.6 0.351 31 0

2 8 183 64 0 0 23.3 0.672 32 1

The difference between above used three approaches for loading CSV data file can easily be understood with the help of given examples.

ML Algorithm Implementation:-

Algorithm for Classification:-

## Introduction to Logistic Regression

Logistic regression is a supervised learning classification algorithm used to predict the probability of a target variable. The nature of target or dependent variable is dichotomous, which means there would be only two possible classes.

In simple words, the dependent variable is binary in nature having data coded as either 1 (stands for success/yes) or 0 (stands for failure/no).

Mathematically, a logistic regression model predicts P(Y=1) as a function of X. It is one of the simplest ML algorithms that can be used for various classification problems such as spam detection, Diabetes prediction, cancer detection etc.

## Types of Logistic Regression

Generally, logistic regression means binary logistic regression having binary target variables, but there can be two more categories of target variables that can be predicted by it. Based on those number of categories, Logistic regression can be divided into following types −

### Binary or Binomial

In such a kind of classification, a dependent variable will have only two possible types either 1 and 0. For example, these variables may represent success or failure, yes or no, win or loss etc.

### Multinomial

In such a kind of classification, dependent variable can have 3 or more possible unordered types or the types having no quantitative significance. For example, these variables may represent “Type A” or “Type B” or “Type C”.

### Ordinal

In such a kind of classification, dependent variable can have 3 or more possible ordered types or the types having a quantitative significance. For example, these variables may represent “poor” or “good”, “very good”, “Excellent” and each category can have the scores like 0,1,2,3.

## Logistic Regression Assumptions

Before diving into the implementation of logistic regression, we must be aware of the following assumptions about the same −

* In case of binary logistic regression, the target variables must be binary always and the desired outcome is represented by the factor level 1.
* There should not be any multi-collinearity in the model, which means the independent variables must be independent of each other .
* We must include meaningful variables in our model.
* We should choose a large sample size for logistic regression.

## Binary Logistic Regression model

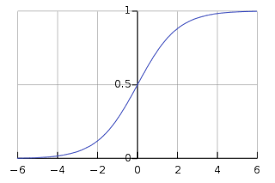
The simplest form of logistic regression is binary or binomial logistic regression in which the target or dependent variable can have only 2 possible types either 1 or 0. It allows us to model a relationship between multiple predictor variables and a binary/binomial target variable. In case of logistic regression, the linear function is basically used as an input to another function such as 𝑔 in the following relation −

hθ(x)=g(θTx)where0≤hθ≤1hθ(x)=g(θTx)𝑤ℎ𝑒𝑟𝑒0≤hθ≤1

Here, 𝑔 is the logistic or sigmoid function which can be given as follows −

g(z)=11+e−zwherez=θTxg(z)=11+e−z𝑤ℎ𝑒𝑟𝑒𝑧=θT𝑥

To sigmoid curve can be represented with the help of following graph. We can see the values of y-axis lie between 0 and 1 and crosses the axis at 0.5.



The classes can be divided into positive or negative. The output comes under the probability of positive class if it lies between 0 and 1. For our implementation, we are interpreting the output of hypothesis function as positive if it is ≥0.5, otherwise negative.

We also need to define a loss function to measure how well the algorithm performs using the weights on functions, represented by theta as follows −

ℎ=𝑔(𝑋𝜃)

J(θ)=1m.(−yTlog(h)−(1−y)Tlog(1−h))J(θ)=1m.(−yTlog(h)−(1−y)Tlog(1−h))

Now, after defining the loss function our prime goal is to minimize the loss function. It can be done with the help of fitting the weights which means by increasing or decreasing the weights. With the help of derivatives of the loss function w.r.t each weight, we would be able to know what parameters should have high weight and what should have smaller weight.

The following gradient descent equation tells us how loss would change if we modified the parameters −

δJ(θ)δθj=1mXT(g(Xθ)−y)𝛿𝐽(𝜃)𝛿θj=1mXT(𝑔(𝑋𝜃)−𝑦)

## Implementation in Python

Now we will implement the above concept of binomial logistic regression in Python. For this purpose, we are using a multivariate flower dataset named ‘iris’ which have 3 classes of 50 instances each, but we will be using the first two feature columns. Every class represents a type of iris flower.

First, we need to import the necessary libraries as follows −

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn import datasets

Next, load the iris dataset as follows −

iris = datasets.load\_iris()

X = iris.data[:, :2]

y = (iris.target != 0) \* 1

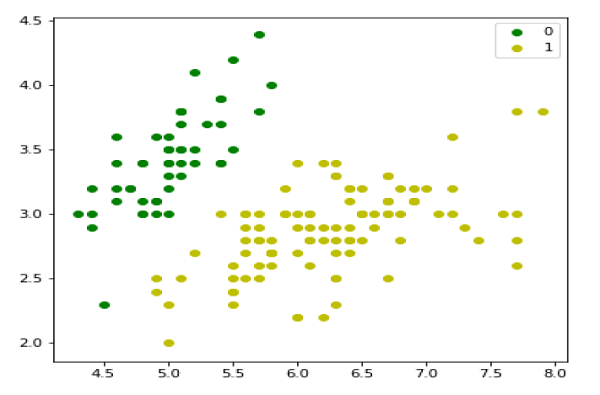
We can plot our training data s follows −

plt.figure(figsize=(6, 6))

plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1], color='g', label='0')

plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='y', label='1')

plt.legend();



Next, we will define sigmoid function, loss function and gradient descend as follows −

class LogisticRegression:

def \_\_init\_\_(self, lr=0.01, num\_iter=100000, fit\_intercept=True, verbose=False):

self.lr = lr

self.num\_iter = num\_iter

self.fit\_intercept = fit\_intercept

self.verbose = verbose

def \_\_add\_intercept(self, X):

intercept = np.ones((X.shape[0], 1))

return np.concatenate((intercept, X), axis=1)

def \_\_sigmoid(self, z):

return 1 / (1 + np.exp(-z))

def \_\_loss(self, h, y):

return (-y \* np.log(h) - (1 - y) \* np.log(1 - h)).mean()

def fit(self, X, y):

if self.fit\_intercept:

X = self.\_\_add\_intercept(X)

Now, initialize the weights as follows −

self.theta = np.zeros(X.shape[1])

for i in range(self.num\_iter):

z = np.dot(X, self.theta)

h = self.\_\_sigmoid(z)

gradient = np.dot(X.T, (h - y)) / y.size

self.theta -= self.lr \* gradient

z = np.dot(X, self.theta)

h = self.\_\_sigmoid(z)

loss = self.\_\_loss(h, y)

if(self.verbose ==True and i % 10000 == 0):

print(f'loss: {loss} \t')

With the help of the following script, we can predict the output probabilities −

def predict\_prob(self, X):

if self.fit\_intercept:

X = self.\_\_add\_intercept(X)

return self.\_\_sigmoid(np.dot(X, self.theta))

def predict(self, X):

return self.predict\_prob(X).round()

Next, we can evaluate the model and plot it as follows −

model = LogisticRegression(lr=0.1, num\_iter=300000)

preds = model.predict(X)

(preds == y).mean()

plt.figure(figsize=(10, 6))

plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1], color='g', label='0')

plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='y', label='1')

plt.legend()

x1\_min, x1\_max = X[:,0].min(), X[:,0].max(),

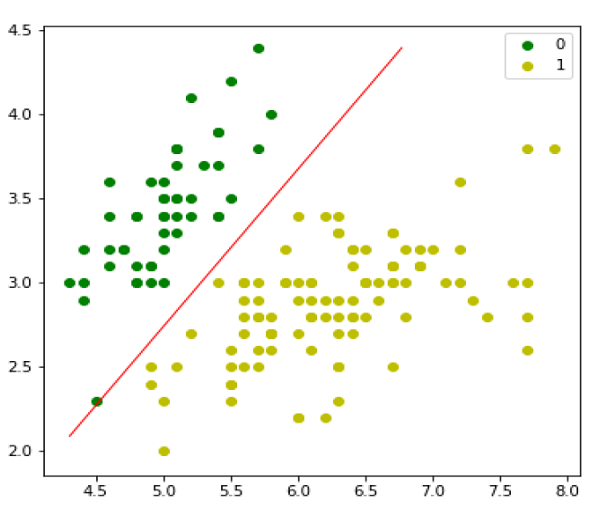
x2\_min, x2\_max = X[:,1].min(), X[:,1].max(),

xx1, xx2 = np.meshgrid(np.linspace(x1\_min, x1\_max), np.linspace(x2\_min, x2\_max))

grid = np.c\_[xx1.ravel(), xx2.ravel()]

probs = model.predict\_prob(grid).reshape(xx1.shape)

plt.contour(xx1, xx2, probs, [0.5], linewidths=1, colors='red');



## Multinomial Logistic Regression Model

Another useful form of logistic regression is multinomial logistic regression in which the target or dependent variable can have 3 or more possible unordered types i.e. the types having no quantitative significance.

## Implementation in Python

Now we will implement the above concept of multinomial logistic regression in Python. For this purpose, we are using a dataset from sklearn named digit.

First, we need to import the necessary libraries as follows −

Import sklearn

from sklearn import datasets

from sklearn import linear\_model

from sklearn import metrics

from sklearn.model\_selection import train\_test\_split

Next, we need to load digit dataset −

digits = datasets.load\_digits()

Now, define the feature matrix(X) and response vector(y)as follows −

X = digits.data

y = digits.target

With the help of next line of code, we can split X and y into training and testing sets −

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4, random\_state=1)

Now create an object of logistic regression as follows −

digreg = linear\_model.LogisticRegression()

Now, we need to train the model by using the training sets as follows −

digreg.fit(X\_train, y\_train)

Next, make the predictions on testing set as follows −

y\_pred = digreg.predict(X\_test)

Next print the accuracy of the model as follows −

print("Accuracy of Logistic Regression model is:",

metrics.accuracy\_score(y\_test, y\_pred)\*100)

### Output

Accuracy of Logistic Regression model is: 95.6884561891516

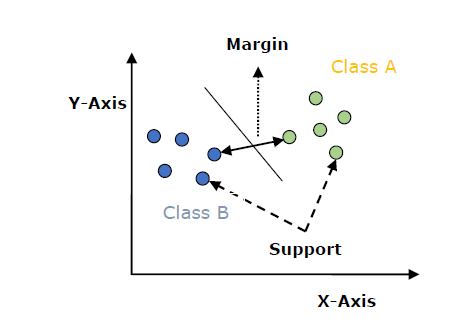
From the above output we can see the accuracy of our model is around 96 percent.

## Introduction to SVM

Support vector machines (SVMs) are powerful yet flexible supervised machine learning algorithms which are used both for classification and regression. But generally, they are used in classification problems. In 1960s, SVMs were first introduced but later they got refined in 1990. SVMs have their unique way of implementation as compared to other machine learning algorithms. Lately, they are extremely popular because of their ability to handle multiple continuous and categorical variables.

## Working of SVM

An SVM model is basically a representation of different classes in a hyperplane in multidimensional space. The hyperplane will be generated in an iterative manner by SVM so that the error can be minimized. The goal of SVM is to divide the datasets into classes to find a maximum marginal hyperplane (MMH).



The followings are important concepts in SVM −

* **Support Vectors** − Datapoints that are closest to the hyperplane is called support vectors. Separating line will be defined with the help of these data points.
* **Hyperplane** − As we can see in the above diagram, it is a decision plane or space which is divided between a set of objects having different classes.
* **Margin** − It may be defined as the gap between two lines on the closet data points of different classes. It can be calculated as the perpendicular distance from the line to the support vectors. Large margin is considered as a good margin and small margin is considered as a bad margin.

The main goal of SVM is to divide the datasets into classes to find a maximum marginal hyperplane (MMH) and it can be done in the following two steps −

* First, SVM will generate hyperplanes iteratively that segregates the classes in best way.
* Then, it will choose the hyperplane that separates the classes correctly.

## Implementing SVM in Python

For implementing SVM in Python we will start with the standard libraries import as follows −

import numpy as np

import matplotlib.pyplot as plt

from scipy import stats

import seaborn as sns; sns.set()

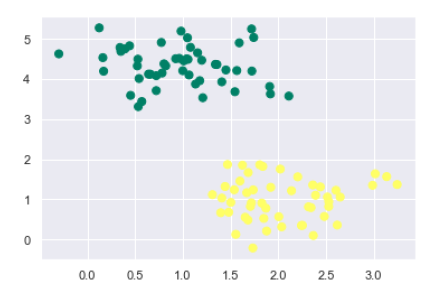
Next, we are creating a sample dataset, having linearly separable data, from sklearn.dataset.sample\_generator for classification using SVM −

from sklearn.datasets.samples\_generator import make\_blobs

X, y = make\_blobs(n\_samples=100, centers=2, random\_state=0, cluster\_std=0.50)

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='summer');

The following would be the output after generating sample dataset having 100 samples and 2 clusters −



We know that SVM supports discriminative classification. it divides the classes from each other by simply finding a line in case of two dimensions or manifold in case of multiple dimensions. It is implemented on the above dataset as follows −

xfit = np.linspace(-1, 3.5)

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='summer')

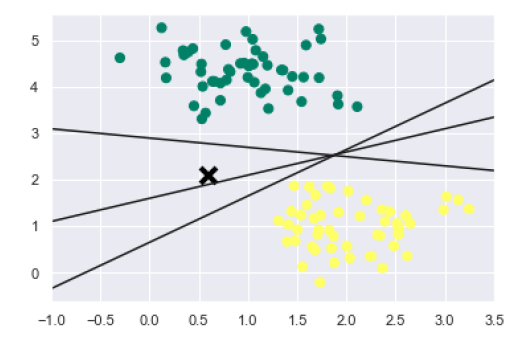
plt.plot([0.6], [2.1], 'x', color='black', markeredgewidth=4, markersize=12)

for m, b in [(1, 0.65), (0.5, 1.6), (-0.2, 2.9)]:

plt.plot(xfit, m \* xfit + b, '-k')

plt.xlim(-1, 3.5);

The output is as follows −



We can see from the above output that there are three different separators that perfectly discriminate the above samples.

As discussed, the main goal of SVM is to divide the datasets into classes to find a maximum marginal hyperplane (MMH) hence rather than drawing a zero line between classes we can draw around each line a margin of some width up to the nearest point. It can be done as follows −

xfit = np.linspace(-1, 3.5)

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='summer')

for m, b, d in [(1, 0.65, 0.33), (0.5, 1.6, 0.55), (-0.2, 2.9, 0.2)]:

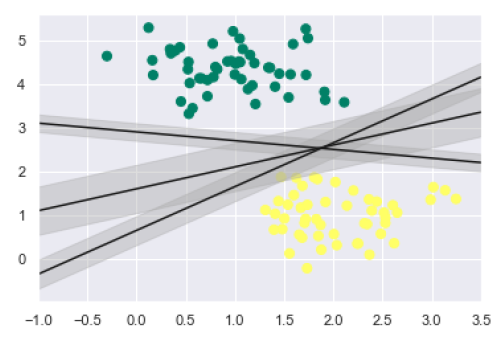
yfit = m \* xfit + b

plt.plot(xfit, yfit, '-k')

plt.fill\_between(xfit, yfit - d, yfit + d, edgecolor='none',

color='#AAAAAA', alpha=0.4)

plt.xlim(-1, 3.5);



From the above image in output, we can easily observe the “margins” within the discriminative classifiers. SVM will choose the line that maximizes the margin.

Next, we will use Scikit-Learn’s support vector classifier to train an SVM model on this data. Here, we are using linear kernel to fit SVM as follows −

from sklearn.svm import SVC # "Support vector classifier"

model = SVC(kernel='linear', C=1E10)

model.fit(X, y)

The output is as follows −

SVC(C=10000000000.0, 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=None,

shrinking=True, tol=0.001, verbose=False)

Now, for a better understanding, the following will plot the decision functions for 2D SVC −

def decision\_function(model, ax=None, plot\_support=True):

if ax is None:

ax = plt.gca()

xlim = ax.get\_xlim()

ylim = ax.get\_ylim()

For evaluating model, we need to create grid as follows −

x = np.linspace(xlim[0], xlim[1], 30)

y = np.linspace(ylim[0], ylim[1], 30)

Y, X = np.meshgrid(y, x)

xy = np.vstack([X.ravel(), Y.ravel()]).T

P = model.decision\_function(xy).reshape(X.shape)

Next, we need to plot decision boundaries and margins as follows −

ax.contour(X, Y, P, colors='k',

levels=[-1, 0, 1], alpha=0.5,

linestyles=['--', '-', '--'])

Now, similarly plot the support vectors as follows −

if plot\_support:

ax.scatter(model.support\_vectors\_[:, 0],

model.support\_vectors\_[:, 1],

s=300, linewidth=1, facecolors='none');

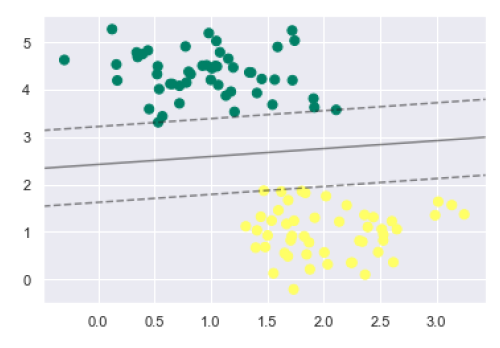
ax.set\_xlim(xlim)

ax.set\_ylim(ylim)

Now, use this function to fit our models as follows −

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='summer')

decision\_function(model);



We can observe from the above output that an SVM classifier fit to the data with margins i.e. dashed lines and support vectors, the pivotal elements of this fit, touching the dashed line. These support vector points are stored in the support\_vectors\_ attribute of the classifier as follows −

model.support\_vectors\_

The output is as follows −

array([[0.5323772 , 3.31338909],

[2.11114739, 3.57660449],

[1.46870582, 1.86947425]])

## SVM Kernels

In practice, SVM algorithm is implemented with kernel that transforms an input data space into the required form. SVM uses a technique called the kernel trick in which kernel takes a low dimensional input space and transforms it into a higher dimensional space. In simple words, kernel converts non-separable problems into separable problems by adding more dimensions to it. It makes SVM more powerful, flexible and accurate. The following are some of the types of kernels used by SVM −

### Linear Kernel

It can be used as a dot product between any two observations. The formula of linear kernel is as below −

k(x,xi) = sum(x\*xi)

From the above formula, we can see that the product between two vectors say 𝑥 & 𝑥𝑖 is the sum of the multiplication of each pair of input values.

### Polynomial Kernel

It is more generalized form of linear kernel and distinguish curved or nonlinear input space. Following is the formula for polynomial kernel −

K(x, xi) = 1 + sum(x \* xi)^d

Here d is the degree of polynomial, which we need to specify manually in the learning algorithm.

### Radial Basis Function (RBF) Kernel

RBF kernel, mostly used in SVM classification, maps input space in indefinite dimensional space. Following formula explains it mathematically −

K(x,xi) = exp(-gamma \* sum((x – xi^2))

Here, gamma ranges from 0 to 1. We need to manually specify it in the learning algorithm. A good default value of gamma is 0.1.

As we implemented SVM for linearly separable data, we can implement it in Python for the data that is not linearly separable. It can be done by using kernels.

### Example

The following is an example for creating an SVM classifier by using kernels. We will be using iris dataset from scikit-learn −

We will start by importing following packages −

import pandas as pd

import numpy as np

from sklearn import svm, datasets

import matplotlib.pyplot as plt

Now, we need to load the input data −

iris = datasets.load\_iris()

From this dataset, we are taking first two features as follows −

X = iris.data[:, :2]

y = iris.target

Next, we will plot the SVM boundaries with original data as follows −

x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

h = (x\_max / x\_min)/100

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h),

np.arange(y\_min, y\_max, h))

X\_plot = np.c\_[xx.ravel(), yy.ravel()]

Now, we need to provide the value of regularization parameter as follows −

C = 1.0

Next, SVM classifier object can be created as follows −

Svc\_classifier = svm.SVC(kernel='linear', C=C).fit(X, y)

Z = svc\_classifier.predict(X\_plot)

Z = Z.reshape(xx.shape)

plt.figure(figsize=(15, 5))

plt.subplot(121)

plt.contourf(xx, yy, Z, cmap=plt.cm.tab10, alpha=0.3)

plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Set1)

plt.xlabel('Sepal length')

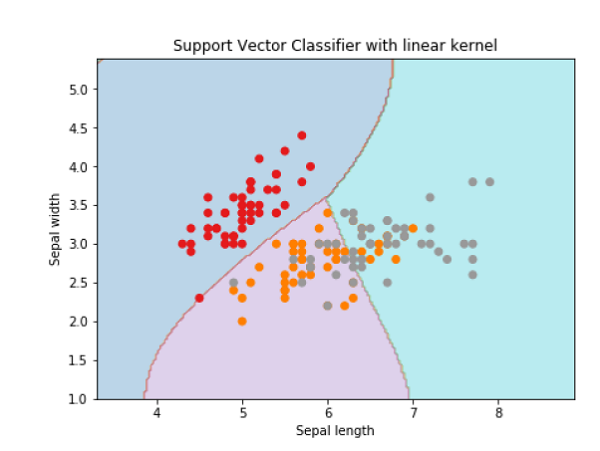
plt.ylabel('Sepal width')

plt.xlim(xx.min(), xx.max())

plt.title('Support Vector Classifier with linear kernel')

### Output

Text(0.5, 1.0, 'Support Vector Classifier with linear kernel')



For creating SVM classifier with **rbf** kernel, we can change the kernel to **rbf** as follows −

Svc\_classifier = svm.SVC(kernel='rbf', gamma =‘auto’,C=C).fit(X, y)

Z = svc\_classifier.predict(X\_plot)

Z = Z.reshape(xx.shape)

plt.figure(figsize=(15, 5))

plt.subplot(121)

plt.contourf(xx, yy, Z, cmap=plt.cm.tab10, alpha=0.3)

plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Set1)

plt.xlabel('Sepal length')

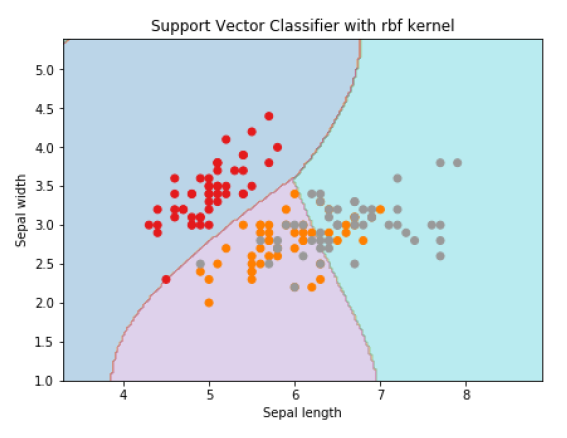
plt.ylabel('Sepal width')

plt.xlim(xx.min(), xx.max())

plt.title('Support Vector Classifier with rbf kernel')

### Output

Text(0.5, 1.0, 'Support Vector Classifier with rbf kernel')



We put the value of gamma to ‘auto’ but you can provide its value between 0 to 1 also.

## Pros and Cons of SVM Classifiers

### Pros of SVM classifiers

SVM classifiers offers great accuracy and work well with high dimensional space. SVM classifiers basically use a subset of training points hence in result uses very less memory.

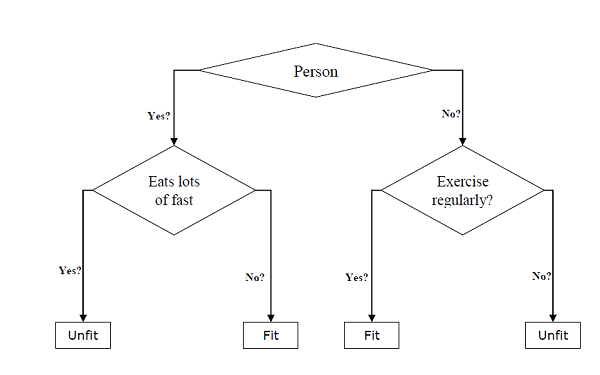
### Cons of SVM classifiers

They have high training time hence in practice not suitable for large datasets. Another disadvantage is that SVM classifiers do not work well with overlapping classes.

## Introduction to Decision Tree

In general, Decision tree analysis is a predictive modelling tool that can be applied across many areas. Decision trees can be constructed by an algorithmic approach that can split the dataset in different ways based on different conditions. Decisions tress are the most powerful algorithms that falls under the category of supervised algorithms.

They can be used for both classification and regression tasks. The two main entities of a tree are decision nodes, where the data is split and leaves, where we got outcome. The example of a binary tree for predicting whether a person is fit or unfit providing various information like age, eating habits and exercise habits, is given below −



In the above decision tree, the question are decision nodes and final outcomes are leaves. We have the following two types of decision trees −

* **Classification decision trees** − In this kind of decision trees, the decision variable is categorical. The above decision tree is an example of classification decision tree.
* **Regression decision trees** − In this kind of decision trees, the decision variable is continuous.

## Implementing Decision Tree Algorithm

### Gini Index

It is the name of the cost function that is used to evaluate the binary splits in the dataset and works with the categorial target variable “Success” or “Failure”.

Higher the value of Gini index, higher the homogeneity. A perfect Gini index value is 0 and worst is 0.5 (for 2 class problem). Gini index for a split can be calculated with the help of following steps −

* First, calculate Gini index for sub-nodes by using the formula p^2+q^2 , which is the sum of the square of probability for success and failure.
* Next, calculate Gini index for split using weighted Gini score of each node of that split.

Classification and Regression Tree (CART) algorithm uses Gini method to generate binary splits.

### Split Creation

A split is basically including an attribute in the dataset and a value. We can create a split in dataset with the help of following three parts −

* **Part1: Calculating Gini Score** − We have just discussed this part in the previous section.
* **Part2: Splitting a dataset** − It may be defined as separating a dataset into two lists of rows having index of an attribute and a split value of that attribute. After getting the two groups - right and left, from the dataset, we can calculate the value of split by using Gini score calculated in first part. Split value will decide in which group the attribute will reside.
* **Part3: Evaluating all splits** − Next part after finding Gini score and splitting dataset is the evaluation of all splits. For this purpose, first, we must check every value associated with each attribute as a candidate split. Then we need to find the best possible split by evaluating the cost of the split. The best split will be used as a node in the decision tree.

## Building a Tree

As we know that a tree has root node and terminal nodes. After creating the root node, we can build the tree by following two parts −

### Part1: Terminal node creation

While creating terminal nodes of decision tree, one important point is to decide when to stop growing tree or creating further terminal nodes. It can be done by using two criteria namely maximum tree depth and minimum node records as follows −

* **Maximum Tree Depth** − As name suggests, this is the maximum number of the nodes in a tree after root node. We must stop adding terminal nodes once a tree reached at maximum depth i.e. once a tree got maximum number of terminal nodes.
* **Minimum Node Records** − It may be defined as the minimum number of training patterns that a given node is responsible for. We must stop adding terminal nodes once tree reached at these minimum node records or below this minimum.

Terminal node is used to make a final prediction.

### Part2: Recursive Splitting

As we understood about when to create terminal nodes, now we can start building our tree. Recursive splitting is a method to build the tree. In this method, once a node is created, we can create the child nodes (nodes added to an existing node) recursively on each group of data, generated by splitting the dataset, by calling the same function again and again.

### Prediction

After building a decision tree, we need to make a prediction about it. Basically, prediction involves navigating the decision tree with the specifically provided row of data.

We can make a prediction with the help of recursive function, as did above. The same prediction routine is called again with the left or the child right nodes.

### Assumptions

The following are some of the assumptions we make while creating decision tree −

* While preparing decision trees, the training set is as root node.
* Decision tree classifier prefers the features values to be categorical. In case if you want to use continuous values then they must be done discretized prior to model building.
* Based on the attribute’s values, the records are recursively distributed.
* Statistical approach will be used to place attributes at any node position i.e.as root node or internal node.

## Implementation in Python

### Example

In the following example, we are going to implement Decision Tree classifier on Pima Indian Diabetes −

First, start with importing necessary python packages −

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split

Next, download the iris dataset from its weblink as follows −

col\_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']

pima = pd.read\_csv(r"C:\pima-indians-diabetes.csv", header=None, names=col\_names)

pima.head()

pregnant glucose bp skin insulin bmi pedigree age label

0 6 148 72 35 0 33.6 0.627 50 1

1 1 85 66 29 0 26.6 0.351 31 0

2 8 183 64 0 0 23.3 0.672 32 1

3 1 89 66 23 94 28.1 0.167 21 0

4 0 137 40 35 168 43.1 2.288 33 1

Now, split the dataset into features and target variable as follows −

feature\_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']

X = pima[feature\_cols] # Features

y = pima.label # Target variable

Next, we will divide the data into train and test split. The following code will split the dataset into 70% training data and 30% of testing data −

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=1)

Next, train the model with the help of DecisionTreeClassifier class of sklearn as follows −

clf = DecisionTreeClassifier()

clf = clf.fit(X\_train,y\_train)

At last we need to make prediction. It can be done with the help of following script −

y\_pred = clf.predict(X\_test)

Next, we can get the accuracy score, confusion matrix and classification report as follows −

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

result = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:")

print(result)

result1 = classification\_report(y\_test, y\_pred)

print("Classification Report:",)

print (result1)

result2 = accuracy\_score(y\_test,y\_pred)

print("Accuracy:",result2)

### Output

Confusion Matrix:

[[116 30]

[ 46 39]]

Classification Report:

precision recall f1-score support

0 0.72 0.79 0.75 146

1 0.57 0.46 0.51 85

micro avg 0.67 0.67 0.67 231

macro avg 0.64 0.63 0.63 231

weighted avg 0.66 0.67 0.66 231

Accuracy: 0.670995670995671

### Visualizing Decision Tree

The above decision tree can be visualized with the help of following code −

from sklearn.tree import export\_graphviz

from sklearn.externals.six import StringIO

from IPython.display import Image

import pydotplus

dot\_data = StringIO()

export\_graphviz(clf, out\_file=dot\_data,

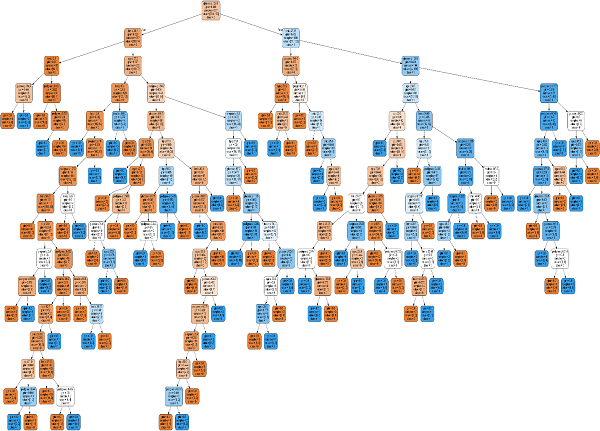
filled=True, rounded=True,

special\_characters=True,feature\_names = feature\_cols,class\_names=['0','1'])

graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue())

graph.write\_png('Pima\_diabetes\_Tree.png')

Image(graph.create\_png())



## Introduction to Naïve Bayes Algorithm

Naïve Bayes algorithms is a classification technique based on applying Bayes’ theorem with a strong assumption that all the predictors are independent to each other. In simple words, the assumption is that the presence of a feature in a class is independent to the presence of any other feature in the same class. For example, a phone may be considered as smart if it is having touch screen, internet facility, good camera etc. Though all these features are dependent on each other, they contribute independently to the probability of that the phone is a smart phone.

In Bayesian classification, the main interest is to find the posterior probabilities i.e. the probability of a label given some observed features, (𝐿 | 𝑓𝑒𝑎𝑡𝑢𝑟𝑒𝑠). With the help of Bayes theorem, we can express this in quantitative form as follows −

P(L|features)=P(L)P(features|L)P(features)P(L|features)=P(L)P(features|L)𝑃(𝑓𝑒𝑎𝑡𝑢𝑟𝑒𝑠)

Here, (𝐿 | 𝑓𝑒𝑎𝑡𝑢𝑟𝑒𝑠) is the posterior probability of class.

(𝐿) is the prior probability of class.

(𝑓𝑒𝑎𝑡𝑢𝑟𝑒𝑠 | 𝐿) is the likelihood which is the probability of predictor given class.

(𝑓𝑒𝑎𝑡𝑢𝑟𝑒𝑠) is the prior probability of predictor.

## Building model using Naïve Bayes in Python

Python library, Scikit learn is the most useful library that helps us to build a Naïve Bayes model in Python. We have the following three types of Naïve Bayes model under Scikit learn Python library −

### Gaussian Naïve Bayes

It is the simplest Naïve Bayes classifier having the assumption that the data from each label is drawn from a simple Gaussian distribution.

### Multinomial Naïve Bayes

Another useful Naïve Bayes classifier is Multinomial Naïve Bayes in which the features are assumed to be drawn from a simple Multinomial distribution. Such kind of Naïve Bayes are most appropriate for the features that represents discrete counts.

### Bernoulli Naïve Bayes

Another important model is Bernoulli Naïve Bayes in which features are assumed to be binary (0s and 1s). Text classification with ‘bag of words’ model can be an application of Bernoulli Naïve Bayes.

### Example

Depending on our data set, we can choose any of the Naïve Bayes model explained above. Here, we are implementing Gaussian Naïve Bayes model in Python −

We will start with required imports as follows −

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns; sns.set()

Now, by using make\_blobs() function of Scikit learn, we can generate blobs of points with Gaussian distribution as follows −

from sklearn.datasets import make\_blobs

X, y = make\_blobs(300, 2, centers=2, random\_state=2, cluster\_std=1.5)

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='summer');

Next, for using GaussianNB model, we need to import and make its object as follows −

from sklearn.naive\_bayes import GaussianNB

model\_GBN = GaussianNB()

model\_GNB.fit(X, y);

Now, we have to do prediction. It can be done after generating some new data as follows −

rng = np.random.RandomState(0)

Xnew = [-6, -14] + [14, 18] \* rng.rand(2000, 2)

ynew = model\_GNB.predict(Xnew)

Next, we are plotting new data to find its boundaries −

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='summer')

lim = plt.axis()

plt.scatter(Xnew[:, 0], Xnew[:, 1], c=ynew, s=20, cmap='summer', alpha=0.1)

plt.axis(lim);

Now, with the help of following line of codes, we can find the posterior probabilities of first and second label −

yprob = model\_GNB.predict\_proba(Xnew)

yprob[-10:].round(3)

### Output

array([[0.998, 0.002],

[1. , 0. ],

[0.987, 0.013],

[1. , 0. ],

[1. , 0. ],

[1. , 0. ],

[1. , 0. ],

[1. , 0. ],

[0. , 1. ],

[0.986, 0.014]]

)

## Pros & Cons

### Pros

The followings are some pros of using Naïve Bayes classifiers −

* Naïve Bayes classification is easy to implement and fast.
* It will converge faster than discriminative models like logistic regression.
* It requires less training data.
* It is highly scalable in nature, or they scale linearly with the number of predictors and data points.
* It can make probabilistic predictions and can handle continuous as well as discrete data.
* Naïve Bayes classification algorithm can be used for binary as well as multi-class classification problems both.

### Cons

The followings are some cons of using Naïve Bayes classifiers −

* One of the most important cons of Naïve Bayes classification is its strong feature independence because in real life it is almost impossible to have a set of features which are completely independent of each other.
* Another issue with Naïve Bayes classification is its ‘zero frequency’ which means that if a categorial variable has a category but not being observed in training data set, then Naïve Bayes model will assign a zero probability to it and it will be unable to make a prediction.

## Applications of Naïve Bayes classification

The following are some common applications of Naïve Bayes classification −

**Real-time prediction** − Due to its ease of implementation and fast computation, it can be used to do prediction in real-time.

**Multi-class prediction** − Naïve Bayes classification algorithm can be used to predict posterior probability of multiple classes of target variable.

**Text classification** − Due to the feature of multi-class prediction, Naïve Bayes classification algorithms are well suited for text classification. That is why it is also used to solve problems like spam-filtering and sentiment analysis.

**Recommendation system** − Along with the algorithms like collaborative filtering, Naïve Bayes makes a Recommendation system which can be used to filter unseen information and to predict weather a user would like the given resource or not.

## Introduction

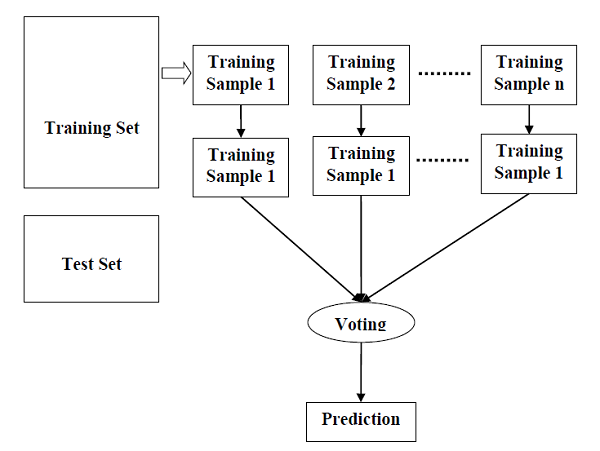
Random forest is a supervised learning algorithm which is used for both classification as well as regression. But however, it is mainly used for classification problems. As we know that a forest is made up of trees and more trees means more robust forest. Similarly, random forest algorithm creates decision trees on data samples and then gets the prediction from each of them and finally selects the best solution by means of voting. It is an ensemble method which is better than a single decision tree because it reduces the over-fitting by averaging the result.

## Working of Random Forest Algorithm

We can understand the working of Random Forest algorithm with the help of following steps −

* **Step 1** − First, start with the selection of random samples from a given dataset.
* **Step 2** − Next, this algorithm will construct a decision tree for every sample. Then it will get the prediction result from every decision tree.
* **Step 3** − In this step, voting will be performed for every predicted result.
* **Step 4** − At last, select the most voted prediction result as the final prediction result.

The following diagram will illustrate its working −



## Implementation in Python

First, start with importing necessary Python packages −

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

Next, download the iris dataset from its weblink as follows −

path = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

Next, we need to assign column names to the dataset as follows −

headernames = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

Now, we need to read dataset to pandas dataframe as follows −

dataset = pd.read\_csv(path, names=headernames)

dataset.head()

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **sepal-length** | **sepal-width** | **petal-length** | **petal-width** | **Class** |
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |

Data Preprocessing will be done with the help of following script lines −

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, 4].values

Next, we will divide the data into train and test split. The following code will split the dataset into 70% training data and 30% of testing data −

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.30)

Next, train the model with the help of RandomForestClassifier class of sklearn as follows −

from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n\_estimators=50)

classifier.fit(X\_train, y\_train)

At last, we need to make prediction. It can be done with the help of following script −

y\_pred = classifier.predict(X\_test)

Next, print the results as follows −

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

result = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:")

print(result)

result1 = classification\_report(y\_test, y\_pred)

print("Classification Report:",)

print (result1)

result2 = accuracy\_score(y\_test,y\_pred)

print("Accuracy:",result2)

### Output

Confusion Matrix:

[

[14 0 0]

[ 0 18 1]

[ 0 0 12]

]

Classification Report:

precision recall f1-score support

Iris-setosa 1.00 1.00 1.00 14

Iris-versicolor 1.00 0.95 0.97 19

Iris-virginica 0.92 1.00 0.96 12

micro avg 0.98 0.98 0.98 45

macro avg 0.97 0.98 0.98 45

weighted avg 0.98 0.98 0.98 45

Accuracy: 0.9777777777777777

## Pros and Cons of Random Forest

### Pros

The following are the advantages of Random Forest algorithm −

* It overcomes the problem of overfitting by averaging or combining the results of different decision trees.
* Random forests work well for a large range of data items than a single decision tree does.
* Random forest has less variance then single decision tree.
* Random forests are very flexible and possess very high accuracy.
* Scaling of data does not require in random forest algorithm. It maintains good accuracy even after providing data without scaling.
* Scaling of data does not require in random forest algorithm. It maintains good accuracy even after providing data without scaling.

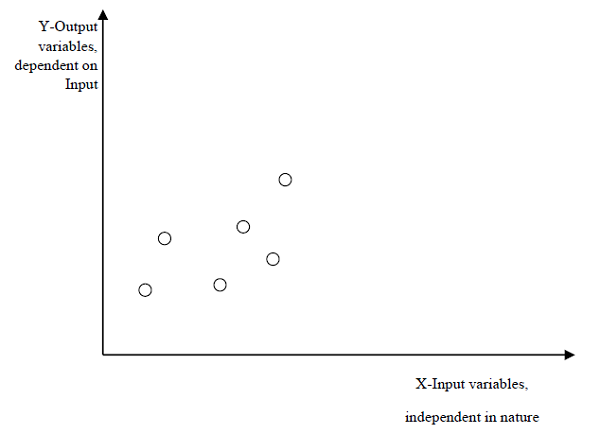
### Cons

The following are the disadvantages of Random Forest algorithm −

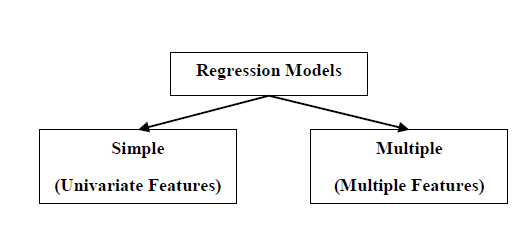
* Complexity is the main disadvantage of Random forest algorithms.
* Construction of Random forests are much harder and time-consuming than decision trees.
* More computational resources are required to implement Random Forest algorithm.
* It is less intuitive in case when we have a large collection of decision trees .
* The prediction process using random forests is very time-consuming in comparison with other algorithms.

## Introduction to Regression

Regression is another important and broadly used statistical and machine learning tool. The key objective of regression-based tasks is to predict output labels or responses which are continues numeric values, for the given input data. The output will be based on what the model has learned in training phase. Basically, regression models use the input data features (independent variables) and their corresponding continuous numeric output values (dependent or outcome variables) to learn specific association between inputs and corresponding outputs.



## Types of Regression Models



Regression models are of following two types −

**Simple regression model** − This is the most basic regression model in which predictions are formed from a single, univariate feature of the data.

**Multiple regression model** − As name implies, in this regression model the predictions are formed from multiple features of the data.

## Building a Regressor in Python

Regressor model in Python can be constructed just like we constructed the classifier. Scikit-learn, a Python library for machine learning can also be used to build a regressor in Python.

In the following example, we will be building basic regression model that will fit a line to the data i.e. linear regressor. The necessary steps for building a regressor in Python are as follows −

### Step 1: Importing necessary python package

For building a regressor using scikit-learn, we need to import it along with other necessary packages. We can import the by using following script −

import numpy as np

from sklearn import linear\_model

import sklearn.metrics as sm

import matplotlib.pyplot as plt

### Step 2: Importing dataset

After importing necessary package, we need a dataset to build regression prediction model. We can import it from sklearn dataset or can use other one as per our requirement. We are going to use our saved input data. We can import it with the help of following script −

input = r'C:\linear.txt'

Next, we need to load this data. We are using np.loadtxt function to load it.

input\_data = np.loadtxt(input, delimiter=',')

X, y = input\_data[:, :-1], input\_data[:, -1]

### Step 3: Organizing data into training & testing sets

As we need to test our model on unseen data hence, we will divide our dataset into two parts: a training set and a test set. The following command will perform it −

training\_samples = int(0.6 \* len(X))

testing\_samples = len(X) - num\_training

X\_train, y\_train = X[:training\_samples], y[:training\_samples]

X\_test, y\_test = X[training\_samples:], y[training\_samples:]

### Step 4: Model evaluation & prediction

After dividing the data into training and testing we need to build the model. We will be using LineaRegression() function of Scikit-learn for this purpose. Following command will create a linear regressor object.

reg\_linear= linear\_model.LinearRegression()

Next, train this model with the training samples as follows −

reg\_linear.fit(X\_train, y\_train)

Now, at last we need to do the prediction with the testing data.

y\_test\_pred = reg\_linear.predict(X\_test)

### Step 5: Plot & visualization

After prediction, we can plot and visualize it with the help of following script −

**Example**

plt.scatter(X\_test, y\_test, color='red')

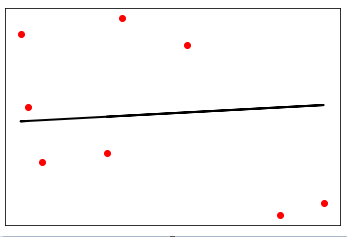
plt.plot(X\_test, y\_test\_pred, color='black', linewidth=2)

plt.xticks(())

plt.yticks(())

plt.show()

**Output**



In the above output, we can see the regression line between the data points.

### Step 6: Performance computation

We can also compute the performance of our regression model with the help of various performance metrics as follows −

**Example**

print("Regressor model performance:")

print("Mean absolute error(MAE) =", round(sm.mean\_absolute\_error(y\_test, y\_test\_pred), 2))

print("Mean squared error(MSE) =", round(sm.mean\_squared\_error(y\_test, y\_test\_pred), 2))

print("Median absolute error =", round(sm.median\_absolute\_error(y\_test, y\_test\_pred), 2))

print("Explain variance score =", round(sm.explained\_variance\_score(y\_test, y\_test\_pred), 2))

print("R2 score =", round(sm.r2\_score(y\_test, y\_test\_pred), 2))

**Output**

Regressor model performance:

Mean absolute error(MAE) = 1.78

Mean squared error(MSE) = 3.89

Median absolute error = 2.01

Explain variance score = -0.09

R2 score = -0.09

## Types of ML Regression Algorithms

The most useful and popular ML regression algorithm is Linear regression algorithm which further divided into two types namely −

* Simple Linear Regression algorithm
* Multiple Linear Regression algorithm.

We will discuss about it and implement it in Python in the next chapter.

## Applications

The applications of ML regression algorithms are as follows −

**Forecasting or Predictive analysis** − One of the important uses of regression is forecasting or predictive analysis. For example, we can forecast GDP, oil prices or in simple words the quantitative data that changes with the passage of time.

**Optimization** − We can optimize business processes with the help of regression. For example, a store manager can create a statistical model to understand the peek time of coming of customers.

**Error correction** − In business, taking correct decision is equally important as optimizing the business process. Regression can help us to take correct decision as well in correcting the already implemented decision.

**Economics** − It is the most used tool in economics. We can use regression to predict supply, demand, consumption, inventory investment etc.

**Finance** − A financial company is always interested in minimizing the risk portfolio and want to know the factors that affects the customers. All these can be predicted with the help of regression model.

## Introduction to Linear Regression

Linear regression may be defined as the statistical model that analyzes the linear relationship between a dependent variable with given set of independent variables. Linear relationship between variables means that when the value of one or more independent variables will change (increase or decrease), the value of dependent variable will also change accordingly (increase or decrease).

Mathematically the relationship can be represented with the help of following equation −

Y = mX + b

Here, Y is the dependent variable we are trying to predict

*X* is the dependent variable we are using to make predictions.

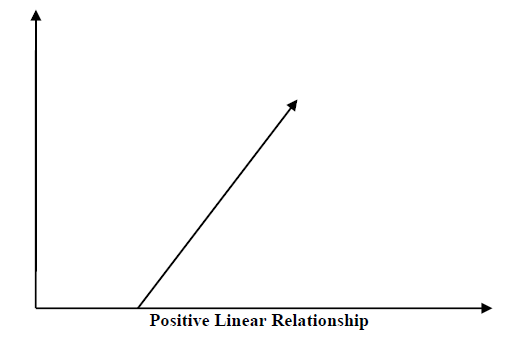
*m* is the slop of the regression line which represents the effect X has on Y

*b* is a constant, known as the Y-intercept. If X = 0,Y would be equal to b.

Furthermore, the linear relationship can be positive or negative in nature as explained below −

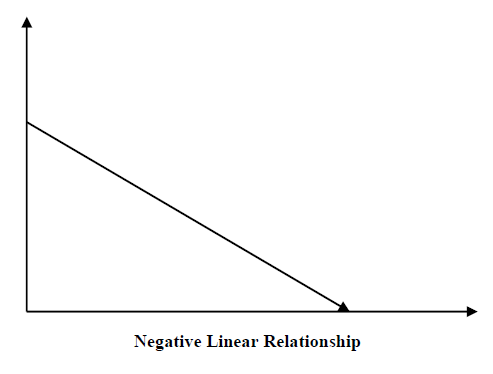
### Positive Linear Relationship

A linear relationship will be called positive if both independent and dependent variable increases. It can be understood with the help of following graph −



### Negative Linear relationship

A linear relationship will be called positive if independent increases and dependent variable decreases. It can be understood with the help of following graph −



## Types of Linear Regression

Linear regression is of the following two types −

* Simple Linear Regression
* Multiple Linear Regression

### Simple Linear Regression (SLR)

It is the most basic version of linear regression which predicts a response using a single feature. The assumption in SLR is that the two variables are linearly related.

### Python implementation

We can implement SLR in Python in two ways, one is to provide your own dataset and other is to use dataset from scikit-learn python library.

**Example 1** − In the following Python implementation example, we are using our own dataset.

First, we will start with importing necessary packages as follows −

%matplotlib inline

import numpy as np

import matplotlib.pyplot as plt

Next, define a function which will calculate the important values for SLR −

def coef\_estimation(x, y):

The following script line will give number of observations n −

n = np.size(x)

The mean of x and y vector can be calculated as follows −

m\_x, m\_y = np.mean(x), np.mean(y)

We can find cross-deviation and deviation about x as follows −

SS\_xy = np.sum(y\*x) - n\*m\_y\*m\_x

SS\_xx = np.sum(x\*x) - n\*m\_x\*m\_x

Next, regression coefficients i.e. b can be calculated as follows −

b\_1 = SS\_xy / SS\_xx

b\_0 = m\_y - b\_1\*m\_x

return(b\_0, b\_1)

Next, we need to define a function which will plot the regression line as well as will predict the response vector −

def plot\_regression\_line(x, y, b):

The following script line will plot the actual points as scatter plot −

plt.scatter(x, y, color = "m", marker = "o", s = 30)

The following script line will predict response vector −

y\_pred = b[0] + b[1]\*x

The following script lines will plot the regression line and will put the labels on them −

plt.plot(x, y\_pred, color = "g")

plt.xlabel('x')

plt.ylabel('y')

plt.show()

At last, we need to define main() function for providing dataset and calling the function we defined above −

def main():

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

y = np.array([100, 300, 350, 500, 750, 800, 850, 900, 1050, 1250])

b = coef\_estimation(x, y)

print("Estimated coefficients:\nb\_0 = {} \nb\_1 = {}".format(b[0], b[1]))

plot\_regression\_line(x, y, b)

if \_\_name\_\_ == "\_\_main\_\_":

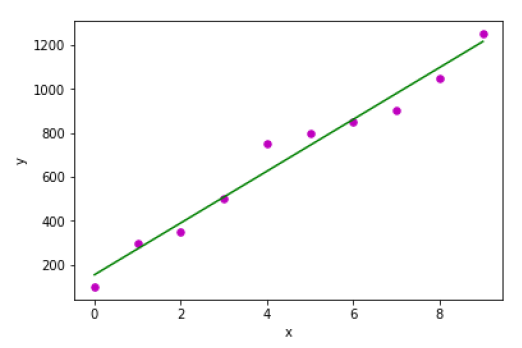
main()

### Output

Estimated coefficients:

b\_0 = 154.5454545454545

b\_1 = 117.87878787878788



**Example 2** − In the following Python implementation example, we are using diabetes dataset from scikit-learn.

First, we will start with importing necessary packages as follows −

%matplotlib inline

import matplotlib.pyplot as plt

import numpy as np

from sklearn import datasets, linear\_model

from sklearn.metrics import mean\_squared\_error, r2\_score

Next, we will load the diabetes dataset and create its object −

diabetes = datasets.load\_diabetes()

As we are implementing SLR, we will be using only one feature as follows −

X = diabetes.data[:, np.newaxis, 2]

Next, we need to split the data into training and testing sets as follows −

X\_train = X[:-30]

X\_test = X[-30:]

Next, we need to split the target into training and testing sets as follows −

y\_train = diabetes.target[:-30]

y\_test = diabetes.target[-30:]

Now, to train the model we need to create linear regression object as follows −

regr = linear\_model.LinearRegression()

Next, train the model using the training sets as follows −

regr.fit(X\_train, y\_train)

Next, make predictions using the testing set as follows −

y\_pred = regr.predict(X\_test)

Next, we will be printing some coefficient like MSE, Variance score etc. as follows −

print('Coefficients: \n', regr.coef\_)

print("Mean squared error: %.2f" % mean\_squared\_error(y\_test, y\_pred))

print('Variance score: %.2f' % r2\_score(y\_test, y\_pred))

Now, plot the outputs as follows −

plt.scatter(X\_test, y\_test, color='blue')

plt.plot(X\_test, y\_pred, color='red', linewidth=3)

plt.xticks(())

plt.yticks(())

plt.show()

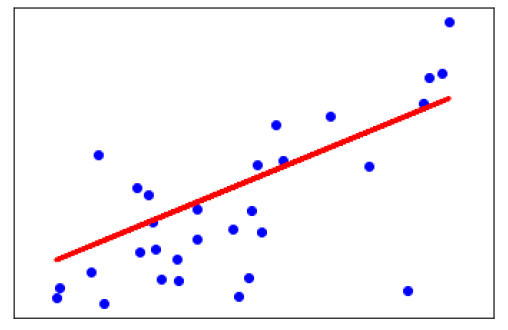
### Output

Coefficients:

[941.43097333]

Mean squared error: 3035.06

Variance score: 0.41



## Multiple Linear Regression (MLR)

It is the extension of simple linear regression that predicts a response using two or more features. Mathematically we can explain it as follows −

Consider a dataset having n observations, p features i.e. independent variables and y as one response i.e. dependent variable the regression line for p features can be calculated as follows −

h(xi)=b0+b1xi1+b2xi2+...+bpxiph(xi)=b0+b1xi1+b2xi2+...+bpxip

Here, h(xi) is the predicted response value and b0,b1,b2…,bp are the regression coefficients.

Multiple Linear Regression models always includes the errors in the data known as residual error which changes the calculation as follows −

h(xi)=b0+b1xi1+b2xi2+...+bpxip+eih(xi)=b0+b1xi1+b2xi2+...+bpxip+ei

We can also write the above equation as follows −

yi=h(xi)+eiorei=yi−h(xi)yi=h(xi)+eiorei=yi−h(xi)

## Python Implementation

in this example, we will be using Boston housing dataset from scikit learn −

First, we will start with importing necessary packages as follows −

%matplotlib inline

import matplotlib.pyplot as plt

import numpy as np

from sklearn import datasets, linear\_model, metrics

Next, load the dataset as follows −

boston = datasets.load\_boston(return\_X\_y=False)

The following script lines will define feature matrix, X and response vector, Y −

X = boston.data

y = boston.target

Next, split the dataset into training and testing sets as follows −

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.7, random\_state=1)

### Example

Now, create linear regression object and train the model as follows −

reg = linear\_model.LinearRegression()

reg.fit(X\_train, y\_train)

print('Coefficients: \n', reg.coef\_)

print('Variance score: {}'.format(reg.score(X\_test, y\_test)))

plt.style.use('fivethirtyeight')

plt.scatter(reg.predict(X\_train), reg.predict(X\_train) - y\_train,

color = "green", s = 10, label = 'Train data')

plt.scatter(reg.predict(X\_test), reg.predict(X\_test) - y\_test,

color = "blue", s = 10, label = 'Test data')

plt.hlines(y = 0, xmin = 0, xmax = 50, linewidth = 2)

plt.legend(loc = 'upper right')

plt.title("Residual errors")

plt.show()

### Output

Coefficients:

[

-1.16358797e-01 6.44549228e-02 1.65416147e-01 1.45101654e+00

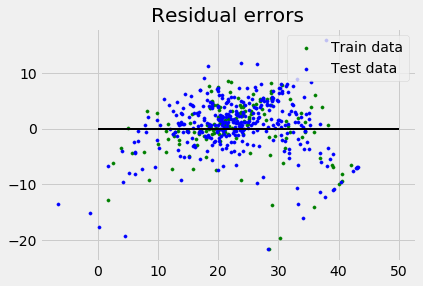
-1.77862563e+01 2.80392779e+00 4.61905315e-02 -1.13518865e+00

3.31725870e-01 -1.01196059e-02 -9.94812678e-01 9.18522056e-03

-7.92395217e-01

]

Variance score: 0.709454060230326



## Assumptions

The following are some assumptions about dataset that is made by Linear Regression model −

**Multi-collinearity** − Linear regression model assumes that there is very little or no multi-collinearity in the data. Basically, multi-collinearity occurs when the independent variables or features have dependency in them.

**Auto-correlation** − Another assumption Linear regression model assumes is that there is very little or no auto-correlation in the data. Basically, auto-correlation occurs when there is dependency between residual errors.

**Relationship between variables** − Linear regression model assumes that the relationship between response and feature variables must be linear.

UNSUPERVISED ML

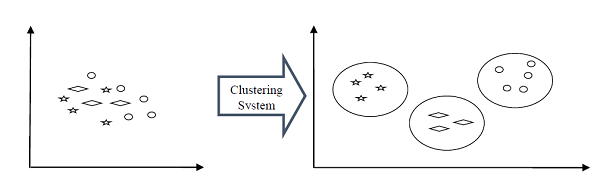
CLUSTERING ALGORTHM:-

## Introduction to Clustering

Clustering methods are one of the most useful unsupervised ML methods. These methods are used to find similarity as well as the relationship patterns among data samples and then cluster those samples into groups having similarity based on features.

Clustering is important because it determines the intrinsic grouping among the present unlabeled data. They basically make some assumptions about data points to constitute their similarity. Each assumption will construct different but equally valid clusters.

For example, below is the diagram which shows clustering system grouped together the similar kind of data in different clusters −



## Cluster Formation Methods

It is not necessary that clusters will be formed in spherical form. Followings are some other cluster formation methods −

### Density-based

In these methods, the clusters are formed as the dense region. The advantage of these methods is that they have good accuracy as well as good ability to merge two clusters. Ex. Density-Based Spatial Clustering of Applications with Noise (DBSCAN), Ordering Points to identify Clustering structure (OPTICS) etc.

### Hierarchical-based

In these methods, the clusters are formed as a tree type structure based on the hierarchy. They have two categories namely, Agglomerative (Bottom up approach) and Divisive (Top down approach). Ex. Clustering using Representatives (CURE), Balanced iterative Reducing Clustering using Hierarchies (BIRCH) etc.

### Partitioning

In these methods, the clusters are formed by portioning the objects into k clusters. Number of clusters will be equal to the number of partitions. Ex. K-means, Clustering Large Applications based upon randomized Search (CLARANS).

### Grid

In these methods, the clusters are formed as a grid like structure. The advantage of these methods is that all the clustering operation done on these grids are fast and independent of the number of data objects. Ex. Statistical Information Grid (STING), Clustering in Quest (CLIQUE).

## Measuring Clustering Performance

One of the most important consideration regarding ML model is assessing its performance or you can say model’s quality. In case of supervised learning algorithms, assessing the quality of our model is easy because we already have labels for every example.

On the other hand, in case of unsupervised learning algorithms we are not that much blessed because we deal with unlabeled data. But still we have some metrics that give the practitioner an insight about the happening of change in clusters depending on algorithm.

Before we deep dive into such metrics, we must understand that these metrics only evaluates the comparative performance of models against each other rather than measuring the validity of the model’s prediction. Followings are some of the metrics that we can deploy on clustering algorithms to measure the quality of model −

## Silhouette Analysis

Silhouette analysis used to check the quality of clustering model by measuring the distance between the clusters. It basically provides us a way to assess the parameters like number of clusters with the help of **Silhouette score**. This score measures how close each point in one cluster is to points in the neighboring clusters.

## Analysis of Silhouette Score

The range of Silhouette score is [-1, 1]. Its analysis is as follows −

* **+1 Score** − Near +1 **Silhouette score** indicates that the sample is far away from its neighboring cluster.
* **0 Score** − 0 **Silhouette score** indicates that the sample is on or very close to the decision boundary separating two neighboring clusters.
* **-1 Score** &minusl -1 **Silhouette score** indicates that the samples have been assigned to the wrong clusters.

The calculation of Silhouette score can be done by using the following formula −

𝒔𝒊𝒍𝒉𝒐𝒖𝒆𝒕𝒕𝒆 𝒔𝒄𝒐𝒓𝒆=(𝒑−𝒒)/𝐦𝐚𝐱 (𝒑,𝒒)

Here, 𝑝 = mean distance to the points in the nearest cluster

And, 𝑞 = mean intra-cluster distance to all the points.

### Davis-Bouldin Index

DB index is another good metric to perform the analysis of clustering algorithms. With the help of DB index, we can understand the following points about clustering model −

* Weather the clusters are well-spaced from each other or not?
* How much dense the clusters are?

We can calculate DB index with the help of following formula −

DB=1n∑i=1nmaxj≠i(σi+σjd(ci,cj))DB=1n∑i=1nmaxj≠i(σi+σjd(ci,cj))

Here, 𝑛 = number of clusters

σi = average distance of all points in cluster 𝑖 from the cluster centroid 𝑐𝑖.

Less the DB index, better the clustering model is.

### Dunn Index

It works same as DB index but there are following points in which both differs −

* The Dunn index considers only the worst case i.e. the clusters that are close together while DB index considers dispersion and separation of all the clusters in clustering model.
* Dunn index increases as the performance increases while DB index gets better when clusters are well-spaced and dense.

We can calculate Dunn index with the help of following formula −

D=min1≤i<j≤nP(i,j)mix1≤i<k≤nq(k)D=min1≤i<j≤nP(i,j)mix1≤i<k≤nq(k)

Here, 𝑖,,𝑘 = each indices for clusters

𝑝 = inter-cluster distance

q = intra-cluster distance

## Types of ML Clustering Algorithms

The following are the most important and useful ML clustering algorithms −

### K-means Clustering

This clustering algorithm computes the centroids and iterates until we it finds optimal centroid. It assumes that the number of clusters are already known. It is also called flat clustering algorithm. The number of clusters identified from data by algorithm is represented by ‘K’ in K-means.

### Mean-Shift Algorithm

It is another powerful clustering algorithm used in unsupervised learning. Unlike K-means clustering, it does not make any assumptions hence it is a non-parametric algorithm.

### Hierarchical Clustering

It is another unsupervised learning algorithm that is used to group together the unlabeled data points having similar characteristics.

We will be discussing all these algorithms in detail in the upcoming chapters.

## Applications of Clustering

We can find clustering useful in the following areas −

**Data summarization and compression** − Clustering is widely used in the areas where we require data summarization, compression and reduction as well. The examples are image processing and vector quantization.

**Collaborative systems and customer segmentation** − Since clustering can be used to find similar products or same kind of users, it can be used in the area of collaborative systems and customer segmentation.

**Serve as a key intermediate step for other data mining tasks** − Cluster analysis can generate a compact summary of data for classification, testing, hypothesis generation; hence, it serves as a key intermediate step for other data mining tasks also.

**Trend detection in dynamic data** − Clustering can also be used for trend detection in dynamic data by making various clusters of similar trends.

**Social network analysis** − Clustering can be used in social network analysis. The examples are generating sequences in images, videos or audios.

**Biological data analysis** − Clustering can also be used to make clusters of images, videos hence it can successfully be used in biological data analysis.

## Introduction to K-Means Algorithm

K-means clustering algorithm computes the centroids and iterates until we it finds optimal centroid. It assumes that the number of clusters are already known. It is also called **flat clustering** algorithm. The number of clusters identified from data by algorithm is represented by ‘K’ in K-means.

In this algorithm, the data points are assigned to a cluster in such a manner that the sum of the squared distance between the data points and centroid would be minimum. It is to be understood that less variation within the clusters will lead to more similar data points within same cluster.

## Working of K-Means Algorithm

We can understand the working of K-Means clustering algorithm with the help of following steps −

* **Step 1** − First, we need to specify the number of clusters, K, need to be generated by this algorithm.
* **Step 2** − Next, randomly select K data points and assign each data point to a cluster. In simple words, classify the data based on the number of data points.
* **Step 3** − Now it will compute the cluster centroids.
* **Step 4** − Next, keep iterating the following until we find optimal centroid which is the assignment of data points to the clusters that are not changing any more −

**4.1** − First, the sum of squared distance between data points and centroids would be computed.

**4.2** − Now, we have to assign each data point to the cluster that is closer than other cluster (centroid).

**4.3** − At last compute the centroids for the clusters by taking the average of all data points of that cluster.

K-means follows **Expectation-Maximization** approach to solve the problem. The Expectation-step is used for assigning the data points to the closest cluster and the Maximization-step is used for computing the centroid of each cluster.

While working with K-means algorithm we need to take care of the following things −

* While working with clustering algorithms including K-Means, it is recommended to standardize the data because such algorithms use distance-based measurement to determine the similarity between data points.
* Due to the iterative nature of K-Means and random initialization of centroids, K-Means may stick in a local optimum and may not converge to global optimum. That is why it is recommended to use different initializations of centroids.

## Implementation in Python

The following two examples of implementing K-Means clustering algorithm will help us in its better understanding −

### Example 1

It is a simple example to understand how k-means works. In this example, we are going to first generate 2D dataset containing 4 different blobs and after that will apply k-means algorithm to see the result.

First, we will start by importing the necessary packages −

%matplotlib inline

import matplotlib.pyplot as plt

import seaborn as sns; sns.set()

import numpy as np

from sklearn.cluster import KMeans

The following code will generate the 2D, containing four blobs −

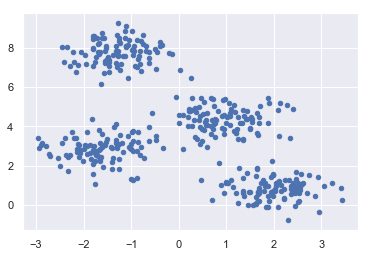
from sklearn.datasets.samples\_generator import make\_blobs

X, y\_true = make\_blobs(n\_samples=400, centers=4, cluster\_std=0.60, random\_state=0)

Next, the following code will help us to visualize the dataset −

plt.scatter(X[:, 0], X[:, 1], s=20);

plt.show()



Next, make an object of KMeans along with providing number of clusters, train the model and do the prediction as follows −

kmeans = KMeans(n\_clusters=4)

kmeans.fit(X)

y\_kmeans = kmeans.predict(X)

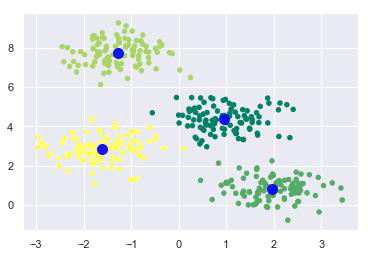
Now, with the help of following code we can plot and visualize the cluster’s centers picked by k-means Python estimator −

plt.scatter(X[:, 0], X[:, 1], c=y\_kmeans, s=20, cmap='summer')

centers = kmeans.cluster\_centers\_

plt.scatter(centers[:, 0], centers[:, 1], c='blue', s=100, alpha=0.9);

plt.show()



### Example 2

Let us move to another example in which we are going to apply K-means clustering on simple digits dataset. K-means will try to identify similar digits without using the original label information.

First, we will start by importing the necessary packages −

%matplotlib inline

import matplotlib.pyplot as plt

import seaborn as sns; sns.set()

import numpy as np

from sklearn.cluster import KMeans

Next, load the digit dataset from sklearn and make an object of it. We can also find number of rows and columns in this dataset as follows −

from sklearn.datasets import load\_digits

digits = load\_digits()

digits.data.shape

### Output

(1797, 64)

The above output shows that this dataset is having 1797 samples with 64 features.

We can perform the clustering as we did in Example 1 above −

kmeans = KMeans(n\_clusters=10, random\_state=0)

clusters = kmeans.fit\_predict(digits.data)

kmeans.cluster\_centers\_.shape

### Output

(10, 64)

The above output shows that K-means created 10 clusters with 64 features.

fig, ax = plt.subplots(2, 5, figsize=(8, 3))

centers = kmeans.cluster\_centers\_.reshape(10, 8, 8)

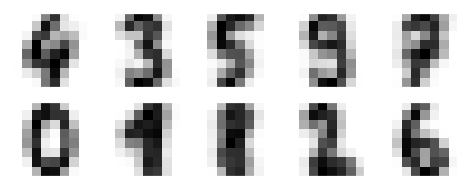
for axi, center in zip(ax.flat, centers):

axi.set(xticks=[], yticks=[])

axi.imshow(center, interpolation='nearest', cmap=plt.cm.binary)

### Output

As output, we will get following image showing clusters centers learned by k-means.



The following lines of code will match the learned cluster labels with the true labels found in them −

from scipy.stats import mode

labels = np.zeros\_like(clusters)

for i in range(10):

mask = (clusters == i)

labels[mask] = mode(digits.target[mask])[0]

Next, we can check the accuracy as follows −

from sklearn.metrics import accuracy\_score

accuracy\_score(digits.target, labels)

### Output

0.7935447968836951

The above output shows that the accuracy is around 80%.

## Advantages and Disadvantages

### Advantages

The following are some advantages of K-Means clustering algorithms −

* It is very easy to understand and implement.
* If we have large number of variables then, K-means would be faster than Hierarchical clustering.
* On re-computation of centroids, an instance can change the cluster.
* Tighter clusters are formed with K-means as compared to Hierarchical clustering.

### Disadvantages

The following are some disadvantages of K-Means clustering algorithms −

* It is a bit difficult to predict the number of clusters i.e. the value of k.
* Output is strongly impacted by initial inputs like number of clusters (value of k).
* Order of data will have strong impact on the final output.
* It is very sensitive to rescaling. If we will rescale our data by means of normalization or standardization, then the output will completely change.final output.
* It is not good in doing clustering job if the clusters have a complicated geometric shape.

## Applications of K-Means Clustering Algorithm

The main goals of cluster analysis are −

* To get a meaningful intuition from the data we are working with.
* Cluster-then-predict where different models will be built for different subgroups.

To fulfill the above-mentioned goals, K-means clustering is performing well enough. It can be used in following applications −

* Market segmentation
* Document Clustering
* Image segmentation
* Image compression
* Customer segmentation
* Analyzing the trend on dynamic data

## Introduction to Mean-Shift Algorithm

As discussed earlier, it is another powerful clustering algorithm used in unsupervised learning. Unlike K-means clustering, it does not make any assumptions; hence it is a non-parametric algorithm.

Mean-shift algorithm basically assigns the datapoints to the clusters iteratively by shifting points towards the highest density of datapoints i.e. cluster centroid.

The difference between K-Means algorithm and Mean-Shift is that later one does not need to specify the number of clusters in advance because the number of clusters will be determined by the algorithm w.r.t data.

## Working of Mean-Shift Algorithm

We can understand the working of Mean-Shift clustering algorithm with the help of following steps −

* **Step 1** − First, start with the data points assigned to a cluster of their own.
* **Step 2** − Next, this algorithm will compute the centroids.
* **Step 3** − In this step, location of new centroids will be updated.
* **Step 4** − Now, the process will be iterated and moved to the higher density region.
* **Step 5** − At last, it will be stopped once the centroids reach at position from where it cannot move further.

## Implementation in Python

It is a simple example to understand how Mean-Shift algorithm works. In this example, we are going to first generate 2D dataset containing 4 different blobs and after that will apply Mean-Shift algorithm to see the result.

%matplotlib inline

import numpy as np

from sklearn.cluster import MeanShift

import matplotlib.pyplot as plt

from matplotlib import style

style.use("ggplot")

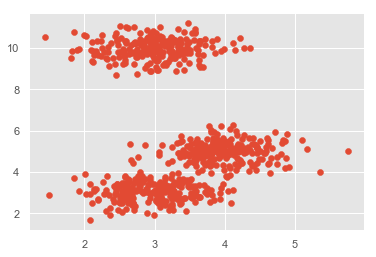
from sklearn.datasets.samples\_generator import make\_blobs

centers = [[3,3,3],[4,5,5],[3,10,10]]

X, \_ = make\_blobs(n\_samples = 700, centers = centers, cluster\_std = 0.5)

plt.scatter(X[:,0],X[:,1])

plt.show()



ms = MeanShift()

ms.fit(X)

labels = ms.labels\_

cluster\_centers = ms.cluster\_centers\_

print(cluster\_centers)

n\_clusters\_ = len(np.unique(labels))

print("Estimated clusters:", n\_clusters\_)

colors = 10\*['r.','g.','b.','c.','k.','y.','m.']

for i in range(len(X)):

plt.plot(X[i][0], X[i][1], colors[labels[i]], markersize = 3)

plt.scatter(cluster\_centers[:,0],cluster\_centers[:,1],

marker=".",color='k', s=20, linewidths = 5, zorder=10)

plt.show()

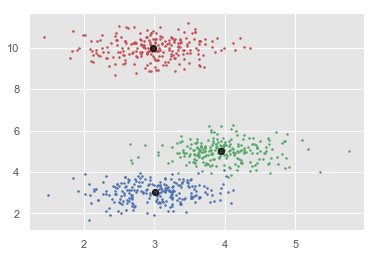
Output

[[ 2.98462798 9.9733794 10.02629344]

[ 3.94758484 4.99122771 4.99349433]

[ 3.00788996 3.03851268 2.99183033]]

Estimated clusters: 3



## Advantages and Disadvantages

### Advantages

The following are some advantages of Mean-Shift clustering algorithm −

* It does not need to make any model assumption as like in K-means or Gaussian mixture.
* It can also model the complex clusters which have nonconvex shape.
* It only needs one parameter named bandwidth which automatically determines the number of clusters.
* There is no issue of local minima as like in K-means.
* No problem generated from outliers.

### Disadvantages

The following are some disadvantages of Mean-Shift clustering algorithm −

Mean-shift algorithm does not work well in case of high dimension, where number of clusters changes abruptly.

* We do not have any direct control on the number of clusters but in some applications, we need a specific number of clusters.
* It cannot differentiate between meaningful and meaningless modes.

## Introduction to Hierarchical Clustering

Hierarchical clustering is another unsupervised learning algorithm that is used to group together the unlabeled data points having similar characteristics. Hierarchical clustering algorithms falls into following two categories −

**Agglomerative hierarchical algorithms** − In agglomerative hierarchical algorithms, each data point is treated as a single cluster and then successively merge or agglomerate (bottom-up approach) the pairs of clusters. The hierarchy of the clusters is represented as a dendrogram or tree structure.

**Divisive hierarchical algorithms** − On the other hand, in divisive hierarchical algorithms, all the data points are treated as one big cluster and the process of clustering involves dividing (Top-down approach) the one big cluster into various small clusters.

## Steps to Perform Agglomerative Hierarchical Clustering

We are going to explain the most used and important Hierarchical clustering i.e. agglomerative. The steps to perform the same is as follows −

* **Step 1** − Treat each data point as single cluster. Hence, we will be having, say K clusters at start. The number of data points will also be K at start.
* **Step 2** − Now, in this step we need to form a big cluster by joining two closet datapoints. This will result in total of K-1 clusters.
* **Step 3** − Now, to form more clusters we need to join two closet clusters. This will result in total of K-2 clusters.
* **Step 4** − Now, to form one big cluster repeat the above three steps until K would become 0 i.e. no more data points left to join.
* **Step 5** − At last, after making one single big cluster, dendrograms will be used to divide into multiple clusters depending upon the problem.

## Role of Dendrograms in Agglomerative Hierarchical Clustering

As we discussed in the last step, the role of dendrogram starts once the big cluster is formed. Dendrogram will be used to split the clusters into multiple cluster of related data points depending upon our problem. It can be understood with the help of following example −

### Example 1

To understand, let us start with importing the required libraries as follows −

%matplotlib inline

import matplotlib.pyplot as plt

import numpy as np

Next, we will be plotting the datapoints we have taken for this example −

X = np.array([[7,8],[12,20],[17,19],[26,15],[32,37],[87,75],[73,85], [62,80],[73,60],[87,96],])

labels = range(1, 11)

plt.figure(figsize=(10, 7))

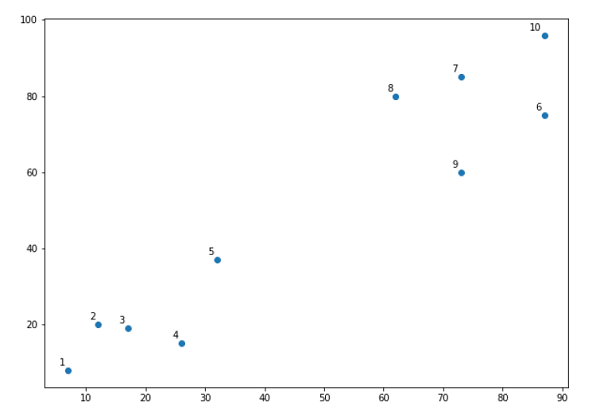
plt.subplots\_adjust(bottom=0.1)

plt.scatter(X[:,0],X[:,1], label='True Position')

for label, x, y in zip(labels, X[:, 0], X[:, 1]):

plt.annotate(label,xy=(x, y), xytext=(-3, 3),textcoords='offset points', ha='right', va='bottom')

plt.show()



From the above diagram, it is very easy to see that we have two clusters in out datapoints but in the real world data, there can be thousands of clusters. Next, we will be plotting the dendrograms of our datapoints by using Scipy library −

from scipy.cluster.hierarchy import dendrogram, linkage

from matplotlib import pyplot as plt

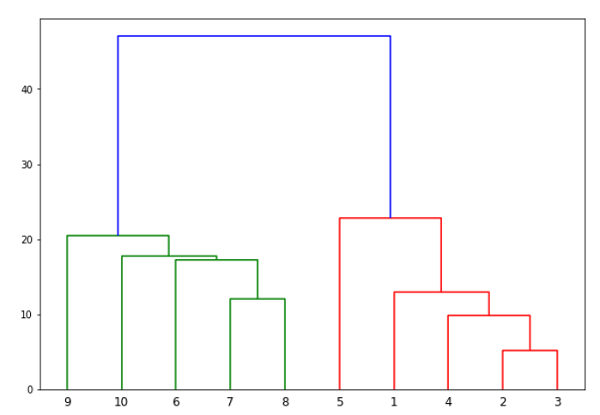
linked = linkage(X, 'single')

labelList = range(1, 11)

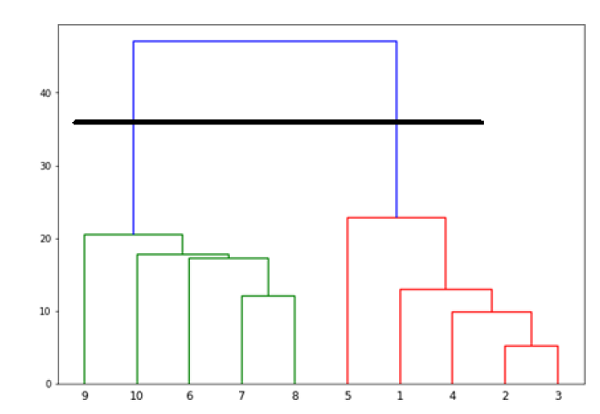
plt.figure(figsize=(10, 7))

dendrogram(linked, orientation='top',labels=labelList, distance\_sort='descending',show\_leaf\_counts=True)

plt.show()



Now, once the big cluster is formed, the longest vertical distance is selected. A vertical line is then drawn through it as shown in the following diagram. As the horizontal line crosses the blue line at two points, the number of clusters would be two.



Next, we need to import the class for clustering and call its fit\_predict method to predict the cluster. We are importing AgglomerativeClustering class of sklearn.cluster library −

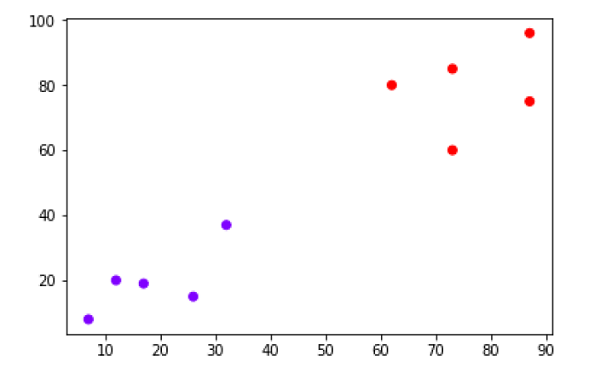
from sklearn.cluster import AgglomerativeClustering

cluster = AgglomerativeClustering(n\_clusters=2, affinity='euclidean', linkage='ward')

cluster.fit\_predict(X)

Next, plot the cluster with the help of following code −

plt.scatter(X[:,0],X[:,1], c=cluster.labels\_, cmap='rainbow')



The above diagram shows the two clusters from our datapoints.

### Example2

As we understood the concept of dendrograms from the simple example discussed above, let us move to another example in which we are creating clusters of the data point in Pima Indian Diabetes Dataset by using hierarchical clustering −

import matplotlib.pyplot as plt

import pandas as pd

%matplotlib inline

import numpy as np

from pandas import read\_csv

path = r"C:\pima-indians-diabetes.csv"

headernames = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

data = read\_csv(path, names=headernames)

array = data.values

X = array[:,0:8]

Y = array[:,8]

data.shape

(768, 9)

data.head()

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **slno.** | **preg** | **Plas** | **Pres** | **skin** | **test** | **mass** | **pedi** | **age** | **class** |
| 0 | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| 1 | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | 0 |
| 2 | 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | 1 |
| 3 | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | 0 |
| 4 | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |

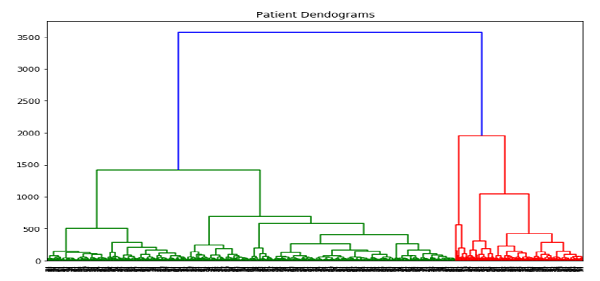
patient\_data = data.iloc[:, 3:5].values

import scipy.cluster.hierarchy as shc

plt.figure(figsize=(10, 7))

plt.title("Patient Dendograms")

dend = shc.dendrogram(shc.linkage(data, method='ward'))



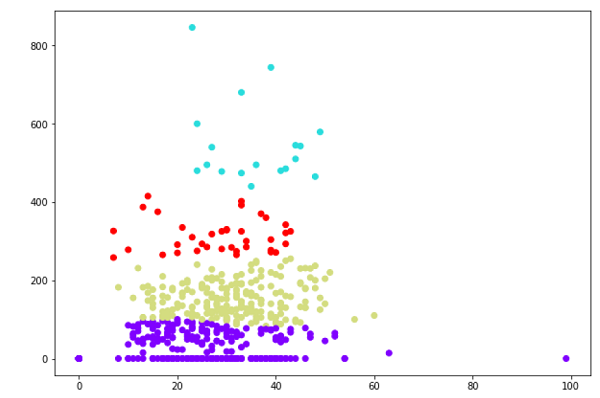
from sklearn.cluster import AgglomerativeClustering

cluster = AgglomerativeClustering(n\_clusters=4, affinity='euclidean', linkage='ward')

cluster.fit\_predict(patient\_data)

plt.figure(figsize=(10, 7))

plt.scatter(patient\_data[:,0], patient\_data[:,1], c=cluster.labels\_, cmap='rainbow')



KNN ALGORITHM:-

## Introduction

K-nearest neighbors (KNN) algorithm is a type of supervised ML algorithm which can be used for both classification as well as regression predictive problems. However, it is mainly used for classification predictive problems in industry. The following two properties would define KNN well −

* **Lazy learning algorithm** − KNN is a lazy learning algorithm because it does not have a specialized training phase and uses all the data for training while classification.
* **Non-parametric learning algorithm** − KNN is also a non-parametric learning algorithm because it doesn’t assume anything about the underlying data.

## Working of KNN Algorithm

K-nearest neighbors (KNN) algorithm uses ‘feature similarity’ to predict the values of new datapoints which further means that the new data point will be assigned a value based on how closely it matches the points in the training set. We can understand its working with the help of following steps −

* **Step 1** − For implementing any algorithm, we need dataset. So during the first step of KNN, we must load the training as well as test data.
* **Step 2** − Next, we need to choose the value of K i.e. the nearest data points. K can be any integer.
* **Step 3** − For each point in the test data do the following −

**3.1** − Calculate the distance between test data and each row of training data with the help of any of the method namely: Euclidean, Manhattan or Hamming distance. The most commonly used method to calculate distance is Euclidean.

**3.2** − Now, based on the distance value, sort them in ascending order.

**3.3** − Next, it will choose the top K rows from the sorted array.

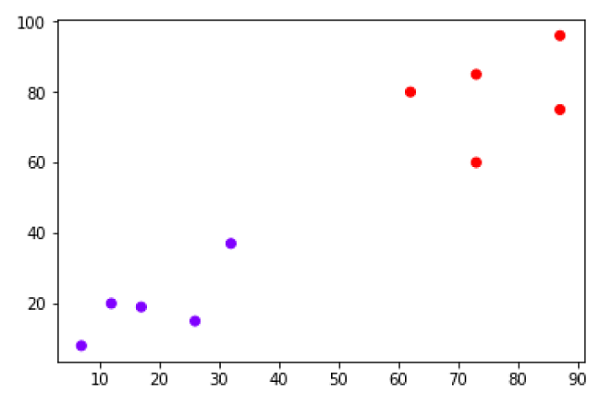
**3.4** − Now, it will assign a class to the test point based on most frequent class of these rows.

* **Step 4** − End

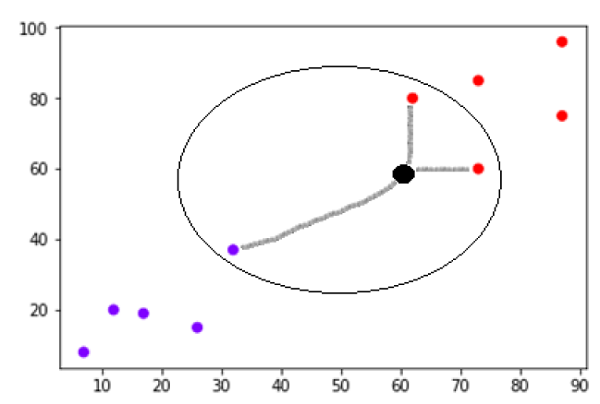
### Example

The following is an example to understand the concept of K and working of KNN algorithm −

Suppose we have a dataset which can be plotted as follows −



Now, we need to classify new data point with black dot (at point 60,60) into blue or red class. We are assuming K = 3 i.e. it would find three nearest data points. It is shown in the next diagram −



We can see in the above diagram the three nearest neighbors of the data point with black dot. Among those three, two of them lies in Red class hence the black dot will also be assigned in red class.

## Implementation in Python

As we know K-nearest neighbors (KNN) algorithm can be used for both classification as well as regression. The following are the recipes in Python to use KNN as classifier as well as regressor −

## KNN as Classifier

First, start with importing necessary python packages −

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

Next, download the iris dataset from its weblink as follows −

path = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

Next, we need to assign column names to the dataset as follows −

headernames = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

Now, we need to read dataset to pandas dataframe as follows −

dataset = pd.read\_csv(path, names=headernames)

dataset.head()

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **slno.** | **sepal-length** | **sepal-width** | **petal-length** | **petal-width** | **Class** |
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |

Data Preprocessing will be done with the help of following script lines −

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, 4].values

Next, we will divide the data into train and test split. Following code will split the dataset into 60% training data and 40% of testing data −

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.40)

Next, data scaling will be done as follows −

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(X\_train)

X\_train = scaler.transform(X\_train)

X\_test = scaler.transform(X\_test)

Next, train the model with the help of KNeighborsClassifier class of sklearn as follows −

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors=8)

classifier.fit(X\_train, y\_train)

At last we need to make prediction. It can be done with the help of following script −

y\_pred = classifier.predict(X\_test)

Next, print the results as follows −

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

result = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:")

print(result)

result1 = classification\_report(y\_test, y\_pred)

print("Classification Report:",)

print (result1)

result2 = accuracy\_score(y\_test,y\_pred)

print("Accuracy:",result2)

### Output

Confusion Matrix:

[[21 0 0]

[ 0 16 0]

[ 0 7 16]]

Classification Report:

precision recall f1-score support

Iris-setosa 1.00 1.00 1.00 21

Iris-versicolor 0.70 1.00 0.82 16

Iris-virginica 1.00 0.70 0.82 23

micro avg 0.88 0.88 0.88 60

macro avg 0.90 0.90 0.88 60

weighted avg 0.92 0.88 0.88 60

Accuracy: 0.8833333333333333

## KNN as Regressor

First, start with importing necessary Python packages −

import numpy as np

import pandas as pd

Next, download the iris dataset from its weblink as follows −

path = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

Next, we need to assign column names to the dataset as follows −

headernames = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

Now, we need to read dataset to pandas dataframe as follows −

data = pd.read\_csv(url, names=headernames)

array = data.values

X = array[:,:2]

Y = array[:,2]

data.shape

output:(150, 5)

Next, import KNeighborsRegressor from sklearn to fit the model −

from sklearn.neighbors import KNeighborsRegressor

knnr = KNeighborsRegressor(n\_neighbors=10)

knnr.fit(X, y)

At last, we can find the MSE as follows −

print ("The MSE is:",format(np.power(y-knnr.predict(X),2).mean()))

### Output

The MSE is: 0.12226666666666669

## Pros and Cons of KNN

### Pros

* It is very simple algorithm to understand and interpret.
* It is very useful for nonlinear data because there is no assumption about data in this algorithm.
* It is a versatile algorithm as we can use it for classification as well as regression.
* It has relatively high accuracy but there are much better supervised learning models than KNN.

### Cons

* It is computationally a bit expensive algorithm because it stores all the training data.
* High memory storage required as compared to other supervised learning algorithms.
* Prediction is slow in case of big N.
* It is very sensitive to the scale of data as well as irrelevant features.

## Applications of KNN

The following are some of the areas in which KNN can be applied successfully −

### Banking System

KNN can be used in banking system to predict weather an individual is fit for loan approval? Does that individual have the characteristics similar to the defaulters one?

### Calculating Credit Ratings

KNN algorithms can be used to find an individual’s credit rating by comparing with the persons having similar traits.

### Politics

With the help of KNN algorithms, we can classify a potential voter into various classes like “Will Vote”, “Will not Vote”, “Will Vote to Party ‘Congress’, “Will Vote to Party ‘BJP’.

Other areas in which KNN algorithm can be used are Speech Recognition, Handwriting Detection, Image Recognition and Video Recognition.