

# **INTRODUCTION TO MACHINE LEARNING**

## TABLE OF CONTENTS

1. INTRODUCTION TO MACHINE LEARNING	1
1.1. INTRODUCTION	1
1.1.1.Supervised Learning	3
1.1.2.Unsupervised Learning	3
1.1.3.Reinforcement Learning	3
1.1.4.Deep Learning	3
1.1.5.Deep Reinforcement Learning	4
2. SUPERVISED MACHINE LEARNING	5
2.1. REGRESSION	6
2.1.1.Linear regression	7
2.1.2.Multiple Linear Regression	10
2.1.3.Polynomial Regression	13
2.2. CLASSIFICATION	13
2.2.1.Challenges in classification	14
2.2.2.Decision Tree Classification	15
2.3. BAYESIAN CLASSIFICATION	23
2.3.1.Conditional probability	23
2.3.2.Independent events	24
2.3.3.Bayes Theorem	24
2.3.4.Naive Bayes algorithm	25
2.4. SUPPORT VECTOR MACHINE (SVM)	29
2.4.1.Hyper plane	34
2.4.2.Two class datasets	34
2.4.3.Maximal margin hyperplanes	35
2.5. K-NEAREST NEIGHBOUR CLASSIFICATION	36
3. UNSUPERVISED MACHINE LEARNING	40
3.1. K-MEANS CLUSTERING	41
3.1.1.K- means clustering algorithm	42
3.2. K-NN (K NEAREST NEIGHBOURS) CLUSTERING	45
3.3. K-MEDOID CLUSTERING OR PAM (PARTITIONING AROUND MEDOIDS)	46
3.4. FUZZY C MEANS CLUSTERING	47
3.5. DBSCAN	51
4. NATURAL LANGUAGE PROCESSING	54
4.1. NATURAL LANGUAGE PROCESSING	54
4.1.1.Types of Natural Language Processing	55
4.1.2.Components of Natural Language Processing (NLP)	56
4.1.3.Stemming	58

4.1.4.Lemmatization	59
4.1.5.Part of Speech Tagging (PoS tagging)	60
4.1.6.Chunking	61
4.1.7.Named Entity Recognition (NER)	63
4.1.8.Bag-of-Words method	63
4.1.9.Term Frequency — Inverse Document Frequency (TF-IDF)	66
4.2. LIBRARIES FOR NATURAL LANGUAGE PROCESSING	69
5. DEEP LEARNING	73
5.1. BIOLOGICAL NEURAL NETWORK	73
5.2. SIMPLE MODEL OF ARTIFICIAL NEURAL NETWORK	75
5.2.1.Types of Activation Functions	76
5.3. TYPES OF ANN	77
5.3.1.Feedforward Network	78
5.3.2.Feedback Network	79
5.3.3.Lateral Network	79
5.4. BACKPROPAGATION	80
5.5. NEED FOR DEEP LEARNING	81
5.5.1.Convolutional Neural Networks	82
5.5.2.Types of CNN	85

# CHAPTER 1

## INTRODUCTION TO MACHINE LEARNING

### 1.1. INTRODUCTION

Artificial Intelligence (AI) refers to the ability of machines to replicate the human intelligence. These machines are well programmed using different algorithms to exhibit human behaviour and actions. The term AI can be simply stated as the intelligence of a machine to learn and solve a problem as humans do. As the availability of computations resources became easy to common man, Artificial Intelligence surpassed the exposure of block chain, virtual reality and quantum computing.

Machine Learning is an application of Artificial Intelligence, Machine Learning can be defined as the ability of a system to learn continuously using the available data and prior experience without being specifically programmed. Creation of such kind of intelligent systems which has the ability to learn themselves needs sufficient data and enormous computational resource. The easy availability of High-Performance Computing (HPC) facilitates the researchers and developers to create various machine learning models which offers better accuracy and results. Cloud providers like Google, Amazon and Microsoft all provides highly configurable GPU ready virtual machines and Machine Learning toolkits which users can leverage to create, train and retrain models.

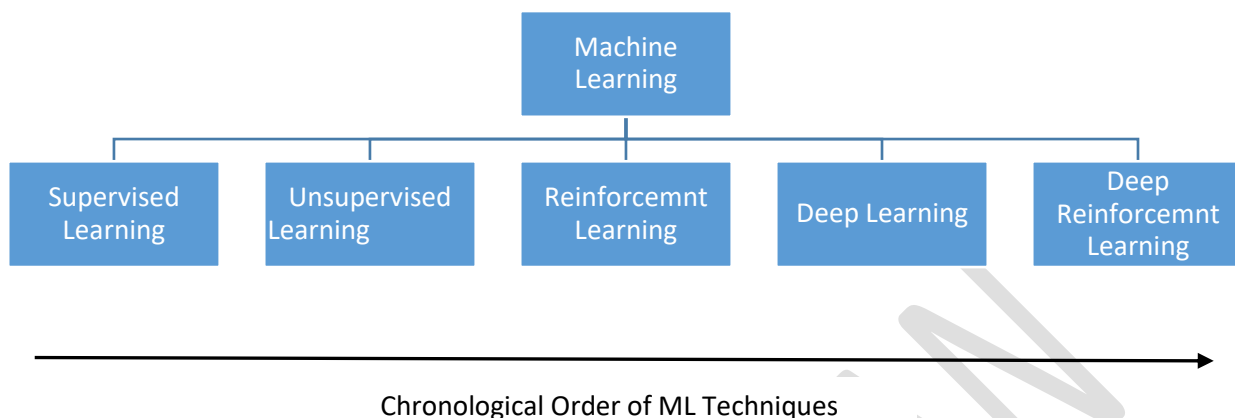
Knowingly or unknowingly we have been using many services which are built up on machine learning models. For example, let's take the scenario of talking to google assistant or Alexa. Both uses complex machine learning models to understand the semantics and context of communication to offer the requested response. They even improve over time by continuously learning your usage. Another example is the smart suggestions on online shopping sites, which displays exactly the same item which you are looking for. This is made possible by studying your past purchase and search preferences using machine learning models. Below listed are few more example scenarios.

## Examples of machine learning

- Real Time Object Identification for self-driving cars
- Language translation & Speech Recognition
- Social Media (E.g.: Friend Suggestions, Face Recognition)
- Search engine result improvements
- Real-time traffic prediction and directions
- Product suggestions in Online shopping sites
- Advanced Email and Malware Threat Protection using ML Models
- Business Intelligence and Stock Market Prediction
- Online fraud detection

The development of today's advanced machine learning started using the traditional statistical techniques such as regression, probability theories, decision tree, classification, clustering. Although machine learning algorithms are developed drastically, an understanding of these techniques are mandatory for your journey through machine learning. Statistical methods were suitable when the availability of data was less. In current world the availability of data is abundant and these techniques have some limitations of their own, hence far more advanced techniques such as deep learning and deep reinforced learning are developed to solve challenging problems.

There are different types of machine learning techniques which are developed as per the increasing requirement of better accuracy and result. They are given in the below chart. The techniques at the far left is the first developed techniques and the right most one is the most advanced.



### 1.1.1. Supervised Learning

Supervised learning is similar to teaching a child to walk. You will hold child's hand, show him how to take his foot forward also you will demonstrate walking to let the child understand and you will continue the same process until the child learns to walk on himself. This technique is the first developed machine technique. In simple you can consider this form of learning where a teacher is present to show the right and wrongs. Regression and Classification techniques are examples of supervised learning.

### 1.1.2. Unsupervised Learning

Unsupervised learning is a type of learning where there is no teacher present, and the system learns without any supervision. Here the system learns based on experience. Clustering is an example of unsupervised learning.

### 1.1.3. Reinforcement Learning

Reinforcement learning introduced when the researchers discovered that It will be nice to provide a reward as the system does the job in intended way. In simple learns here the system learns from mistake.

### 1.1.4. Deep Learning

Neurons in the human brains are simulated as artificial neural networks in our binary computers to solve problems. Such learning techniques is called Deep Learning

### **1.1.5. Deep Reinforcement Learning**

Deep reinforcement learning was introduced by providing rewards to the Artificial Neural Networks.

We will discuss these techniques in detail in the upcoming chapters with sufficient figures and examples. Implementation of these techniques will be done in python with Jupiter notebook interactive development environment. We will be using python packages such as NumPy and scikit-learn for the implementation of ML techniques. Both these packages are open source and easy to integrate to python environment.

## CHAPTER 2

### SUPERVISED MACHINE LEARNING

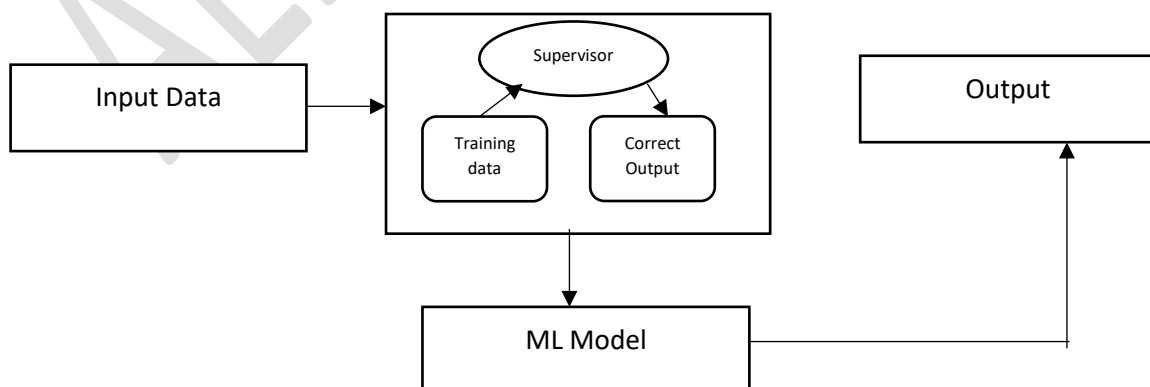
Supervised learning is the first introduced machine learning techniques, which is useful for solving simple problems. In this kind of learning, a teacher is present to show the rights and wrongs and there by system learns.

You can consider supervised learning as a function or a system which maps incoming inputs to outputs based the input output pairs of the training data. Training data normally constitute a number of input output pairs, and the supervised learning algorithms makes a function to map upcoming inputs to outputs based on the training data.

Consider  $x$  as input and  $y$  as output then supervised learning function  $f$  is defined as

$$y = f(x)$$

The presence of the training data in supervised learning can be related to the presence of a teacher for supervising the learning process. The system tries to find the outputs of training data and the teacher corrects it (from training data itself). The learning process continues until the system achieves an acceptable accuracy.





Let us consider one simple example for supervised learning, consider below customer satisfaction table for a beauty clinic.

Gender	Age	Satisfaction
M	45	Unhappy
M	32	Unhappy
F	27	Happy
M	24	Unhappy
F	60	Happy
F	19	Happy

Based on the above data supervised learning algorithm tries to predict the satisfaction of a new customer.

In this section we will discuss the different types of supervised learning algorithms. Regression and classification are the examples of supervised learning algorithms. You must remember that there is no single algorithm which is optimal for all supervised learning problems i.e. the learning model generated for a text classification will not fit properly for image classification problem.

## 2.1. REGRESSION

Regression analysis is one of the most basic and important concepts in statistics and machine learning, regression analysis tries to find relationship among variables. Regression analysis is a supervised learning problem where the system tries to identify output  $y$  from set of input  $x$  and a set of parameters  $\theta$

That is regression function  $g$  defined as

$$y = g(x, \theta)$$

Let's consider one simple example of regression problem, consider the analysis made to understand the salary structure of employees on a particular company by considering the parameters such as experience, position, educational qualification and certifications. Here you can consider  $x$  as the employee,  $\theta$  as the parameters

(experience, position, educational qualification and certifications) and  $y$  as the predicted salary.

Definition: A regression problem is the problem of determining a relation between one or more independent variables and an output variable which is a real continuous variable, given a set of observed values of the set of independent variables and the corresponding values of the output variable.

There are different types of regression based on the type of function ( $g$ ) used to represent the relationship between dependent variable ( $y$ ) and independent variable ( $x$ ). They are Linear regression, Multiple regression and Polynomial regression

### 2.1.1. Linear regression

Liner regression is the simplest form of regression techniques where there is only one independent variable  $x$ . if the relationship between  $x$  and  $y$  defined as

$$y = b + mx$$

then this form of regression is called Liner regression, here  $m$  is called regression coefficient. If we are able to solve  $b$  and  $m$  then it is possible to find  $y$  value for every  $x$  observations and that's how prediction works.

#### Formulas to find $m$ and $b$

These formulae to find  $m$  and  $b$  are obtained by using the methods of calculus

$$m = \frac{Cov(x, y)}{Var(x)}$$

$$b = \bar{y} - m\bar{x}$$

The covariance of  $x$  and  $y$ , denoted by  $Cov(x, y)$  is defined as

$$Cov(x, y) = \frac{1}{n-1} \sum (x_i - \bar{x})(y_i - \bar{y})$$

and also, that the variance of  $x$  is given by

$$Var(x) = \frac{1}{n-1} \sum (x_i - \bar{x})^2$$

Example Problem: Obtain a liner regression for the given data. Consider  $y$  as dependent variable.  $x = [1, 2, 3, 4, 5]$  and  $y = [1, 2, 1.30, 3.75, 2.25]$

*Solution:*

$$n = 5$$

$$\bar{x} = \frac{1 + 2 + 3 + 4 + 5}{5} = 3$$

$$\bar{y} = \frac{1 + 2 + 1.30 + 3.75 + 2.25}{5} = 2.06$$

As per the equations above for covariance and variance, we can find

$$Cov(x, y) = \frac{[(1 - 3)(1 - 2.06) + \dots + (5 - 3)(2.25 - 2.06)]}{4} = 1.0625$$

$$Var(x) = [(1 - 3)^2 + \dots + (5 - 3)^2] = 2.5$$

Applying these values to the equations for  $m$  and  $b$

$$m = \frac{1.0625}{2.5} = 0.425$$

$$b = 2.06 - 0.425 * 3 = 0.785$$

Therefore, the linear regression model for the data is

$$y = 0.785 + 0.425x$$

## **Python Program**

```
# importing required packages and classes, package numpy is imported and class
LinearRegression from sklearn.linear_model

import numpy as np
from sklearn.linear_model import LinearRegression

# passing input x and output y, reshape is done to make x a column matrix, it is
necessary for the LinearRegression to operate
x = np.array([1, 2, 3, 4, 5]).reshape((-1, 1))
y = np.array([1, 2, 1.30, 3.75, 2.25])

# model fitting is performed ie: regression and model and parameters are saved to
the variable model

model = LinearRegression().fit(x, y)
print('b:', model.intercept_) # printing b
print('regression coefficient m:', model.coef_) # printing regression coefficient
```

### **Output:**

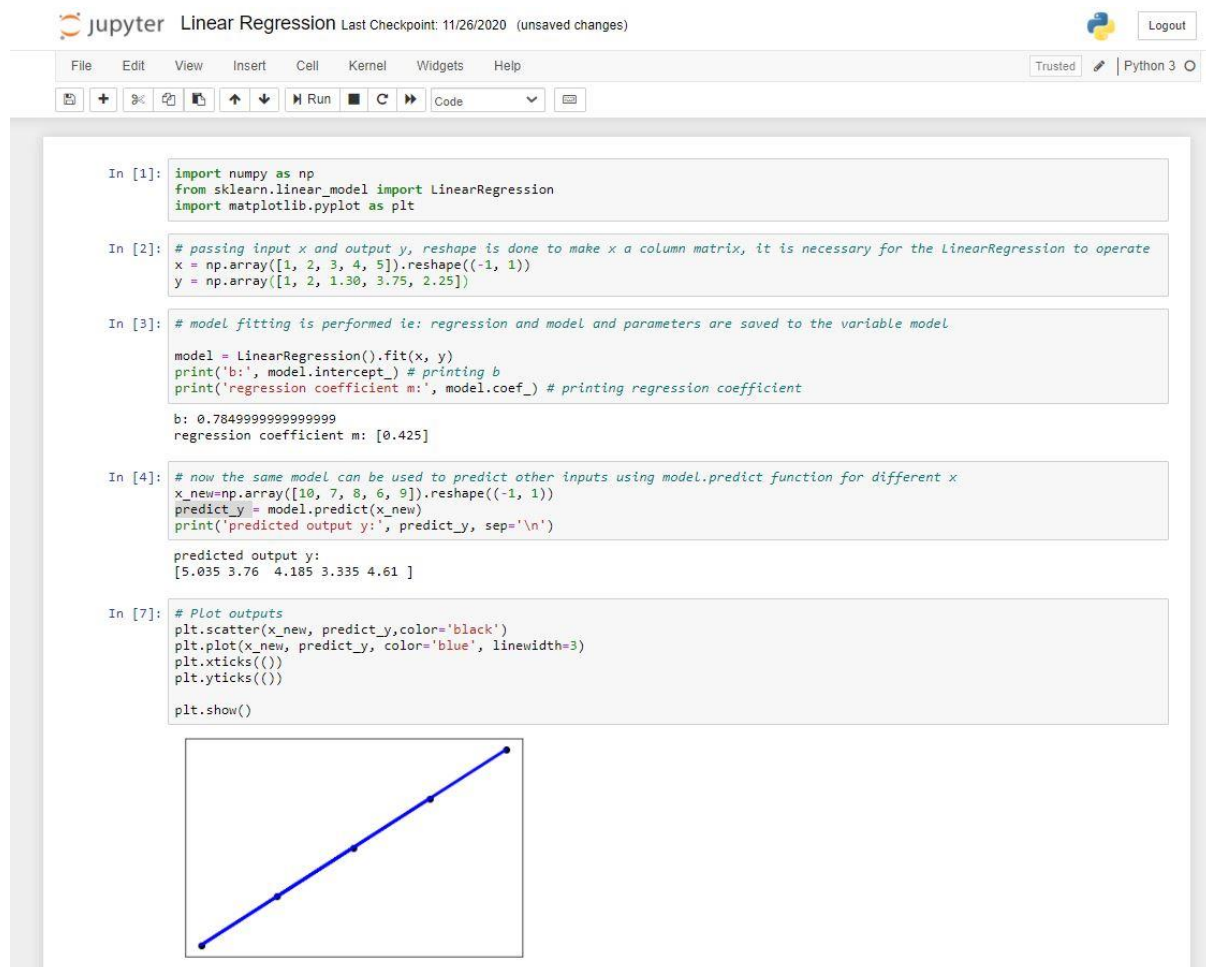
```
b: 0.7849999999999999
regression coefficient m: [0.425]
```

```
# now the same model can be used to predict other inputs using model.predict
function for different x
x_new=np.array([10, 7, 8, 6, 9]).reshape((-1, 1))
predict_y = model.predict(x_new)
print('predicted output y:', predict_y, sep='\n')
```

### **Output:**

```
predicted output y:
[5.035 3.76 4.185 3.335 4.61]
```

## Jupyter Notebook Screenshot



### 2.1.2. Multiple Linear Regression

Multiple regression is the type of regression where there is multiple independent variables such as  $x_1, x_2, x_3 \dots \dots x_n$ . That is the relationship between  $y$  and independent variables models as

$$y = m_0 + m_1x_1 + m_2x_2 + \dots + m_nx_n$$

$m_0, m_1, m_2 \dots \dots m_n$  represents the regression coefficients. Regression coefficients ( $M$ ) need to be found from the known inputs ( $X$ ) and outputs ( $Y$ ) to facilitate prediction.

Let us consider

$$X = \begin{bmatrix} 1 & x_{11} & x_{21} & \dots & x_{N1} \\ 1 & x_{12} & x_{22} & \dots & x_{N2} \\ \vdots & & & & \\ 1 & x_{1n} & x_{2n} & \dots & x_{Nn} \end{bmatrix}$$

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \text{ and } M = \begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ m_N \end{bmatrix}$$

$n$  represent the number of independent variables and  $N$  represents number of values for those variables

Then it is possible to solve the regression coefficients  $M$  using below equation

$$M = (X^T X)^{-1} X^T Y$$

Example: find regression coefficients for the following data

$$X = [(1,1), (1,2), (2,2), (0,1)] \quad Y = [3.25, 6.5, 3.5, 5]$$

Solution: Here we have two independent variable and four set of values for each variable that is  $n = 2$  and  $N = 4$ , and the regression model can be represented as

$$y = m_0 + m_1 x_1 + m_2 x_2$$

Here

$$x_1 = 1, 1, 2, 0 \text{ and } x_2 = 1, 2, 2, 1$$

$$Y = 3.25, 6.5, 3.5, 5$$

As per the representation of  $X, Y$  and  $M$

$$X = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 1 & 2 & 2 \\ 1 & 0 & 1 \end{bmatrix} \quad Y = \begin{bmatrix} 3.25 \\ 6.5 \\ 3.5 \\ 5 \end{bmatrix} \text{ and } M = \begin{bmatrix} m_0 \\ m_1 \\ m_2 \end{bmatrix}$$

$$X^T X = \begin{bmatrix} 4 & 4 & 6 \\ 4 & 6 & 7 \\ 6 & 7 & 10 \end{bmatrix}$$

$$(X^T X)^{-1} = \begin{bmatrix} \frac{11}{4} & \frac{1}{2} & -2 \\ \frac{1}{2} & 1 & -1 \\ -2 & -1 & 2 \end{bmatrix}$$

$$M = (X^T X)^{-1} X^T Y = \begin{bmatrix} 2.0625 \\ -2.3750 \\ 3.2500 \end{bmatrix}$$

Then the required model is  $y = 2.0625 - 2.3750x_1 + 3.2500x_2$

### **Python Program**

# importing required packages and classes, package numpy is imported and class LinearRegression from sklearn.linear\_model. Pass input x and y

```
import numpy as np
from sklearn.linear_model import LinearRegression
x = np.array([[1, 1], [1, 2], [2, 2], [0, 1]])
y = np.array[3.25, 6.5, 3.5, 5]
# model fitting is performed ie: regression and model and parameters are saved to
the variable model
model = LinearRegression().fit(x, y)
print('intercept:', model.intercept_)
print('slope:', model.coef_)
```

#### **Output:**

```
intercept: 2.0625000000000004
slope: [-2.375 3.25]
```

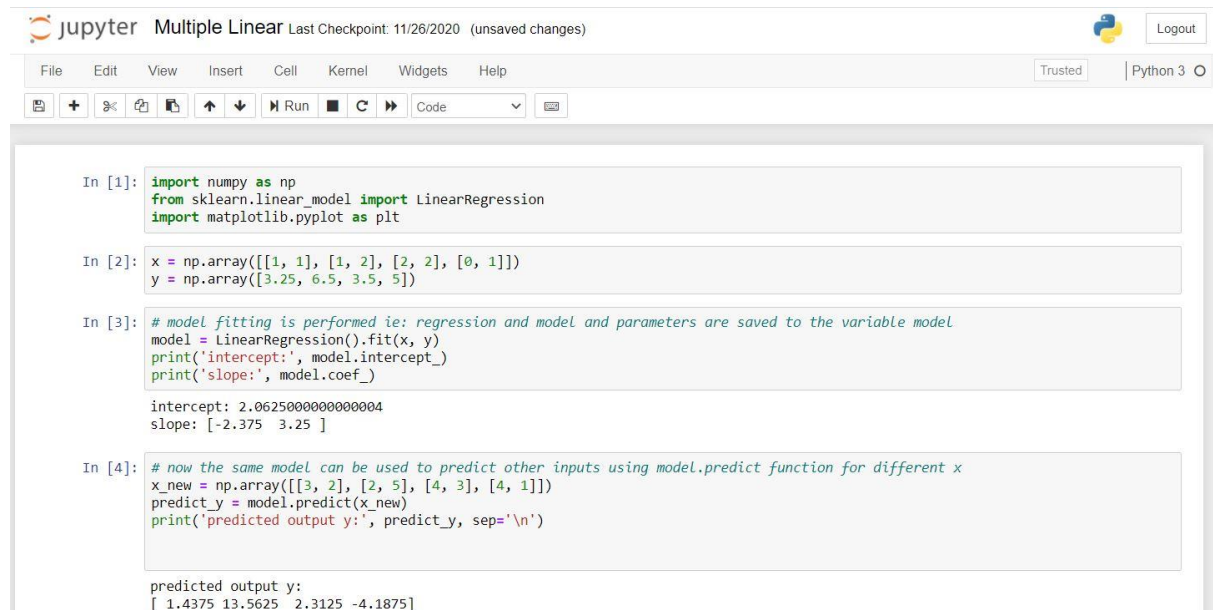
# now the same model can be used to predict other inputs using model.predict function for different x

```
x_new = np.array([[3, 2], [2, 5], [4, 3], [4, 1]])
predict_y = model.predict(x_new)
print('predicted output y:', predict_y, sep='\n')
```

#### **Output:**

```
predicted output y:
[ 1.4375 13.5625 2.3125 -4.1875]
```

## Jupyter Notebook Screenshot



The screenshot shows a Jupyter Notebook titled "Multiple Linear" with a last checkpoint of 11/26/2020. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for saving, running, and other actions. The notebook contains four code cells:

```
In [1]: import numpy as np
        from sklearn.linear_model import LinearRegression
        import matplotlib.pyplot as plt

In [2]: x = np.array([[1, 1], [1, 2], [2, 2], [0, 1]])
        y = np.array([3.25, 6.5, 3.5, 5])

In [3]: # model fitting is performed ie: regression and model and parameters are saved to the variable model
        model = LinearRegression().fit(x, y)
        print('intercept:', model.intercept_)
        print('slope:', model.coef_)

intercept: 2.0625000000000004
slope: [-2.375  3.25 ]

In [4]: # now the same model can be used to predict other inputs using model.predict function for different x
        x_new = np.array([[3, 2], [2, 5], [4, 3], [4, 1]])
        predict_y = model.predict(x_new)
        print('predicted output y:', predict_y, sep='\n')

predicted output y:
[ 1.4375 13.5625  2.3125 -4.1875]
```

### 2.1.3. Polynomial Regression

Consider that there is only one independent variable  $x$  and the relationship between independent variable  $x$  and dependent variable  $y$  is modelled as a polynomial function then such regression is called as polynomial regression.

$$y = m_0 + m_1x + m_2x^2 + \dots + m_nx^n$$

## 2.2. CLASSIFICATION

Classification is simply the process of mapping inputs to categorical labels and a model which performs this task is said to be a classifier. Regression normally applied for numerical problems ie to forecast the turnover for the upcoming business year. Classification problems are quite different and applied to map inputs to corresponding labels. For example, consider a simple activity in a bank, the automated system in the bank can decide to map a loan request from a customer to either "safe" or "risky". Here in this example "safe" and "risky" were the labels.

Classification is a two-step process, in the first step is the learning stage for the model, the model is created with a training set. The training stage includes inputs and their corresponding correct labels. Since the class label of each training inputs is provided, this step is also known as supervised learning (teacher present - correct outputs



known). The Second stage is using the trained model and using it for classifying the new inputs to corresponding labels.

Mathematically you can write a classified as

$$y = f(x)$$

The first stage of classification can be considered as the generation of function  $f$  and the second stage as the operation of function  $f$  on input  $x$  to generate corresponding class label  $y$ . The accuracy of classifier is given as the ratio of correctly classified test samples to total test samples, once a classifier is generated with accepted accuracy then it is used for classification. You can compare different classifiers with the parameters such as accuracy, speed, robustness, and scalability. Accuracy as we explained before it refers to the ability of the given classifier to correctly predict the label, speed refers to the computational resources required to perform the classification, robustness refers to the ability of system to perform when incomplete or noisy data is given and scalability refers to the ability of classifiers to handle large amount of data.

### **2.2.1. Challenges in classification**

Data cleaning: Data cleaning is a pre-processing stage in classification to reduce uncertainty in learning process and thereby to produce a classifier with better accuracy. The input data should be analysed for the presence of noise and absence of valuable data i.e.: missing values. Noise can be treated using several smoothening techniques and missing values can be solved either by replacing that value with a most commonly observed value or by using statistical methods to predict that value.

Relevance analysis: Relevance analysis can be used to identify relevant features that are important for classification. One example for relevance analysis is correlation, if two features are highly correlated, one of those could be removed in the pre-processing stage. Feature subset selection can also be used to reduce the set of features, this technique can be used to find a reduced set of features in a way that the resulting probability distribution of the data classes is as close as possible to the original distribution obtained using all features. Using these two-relevance analysis

techniques, classification problem can be further simplified and made cost effective by removing features that does not contribute to the classification. The presence of such attributes may slow down the whole classifier and could fail during performance comparison against other classifiers.

Data transformation and reduction: Data normalization is a very important pre-processing for classification, for an example consider some features such as distance, or income etc. These normally have large integer values and operation on these large values is complex for complex classification networks or deep networks. Using such data as it could slow down the whole classification process and there will be huge performance loss. In order to address above problem, we will use a technique called normalization. Normalization scales all values for a given feature so that they fall within a small specified range, such as  $-1.0$  to  $1.0$ , or  $0.0$  to  $1.0$ . Normalization applies to the numerical data such as distance, salary and turn over.

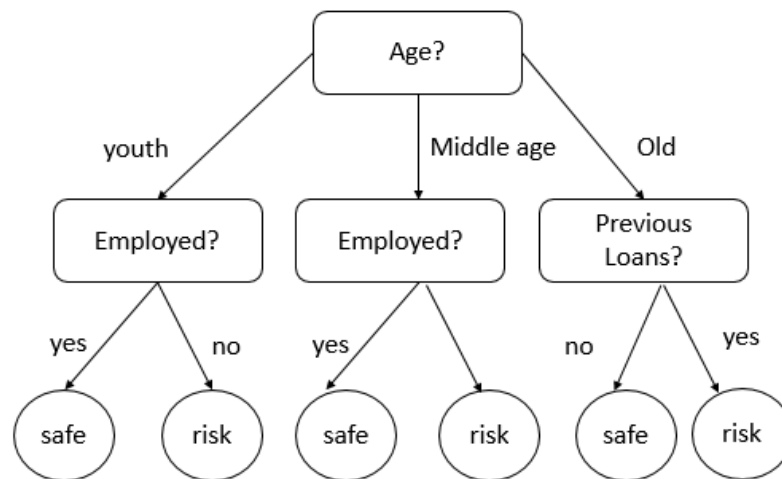
Another data transformation technique is Generalization, data can be generalized to higher level concepts to avoid complexity. For example, a feature marks can be generalized to Grades similarly profit can be generalized to low, medium and high and categorical attributes, like street, can be generalized to higher-level concepts, like city. Generalization process compress the original data to be trained which result in fewer input output operation and leads to less memory utilization and thereby achieve better performance.

There are different types of classification techniques are available such as decision tree classification, Bayesian, Support Vector Machines (SVM), K-Nearest Neighbour classification etc. We will go through these classification techniques in this chapter

### **2.2.2 Decision Tree Classification**

A decision tree is simply a tress structure, where each internal node denotes a question/test on the attribute, each branch represents the possible outcome of that question/test and each leaf node represents the class label. The process of generating decision tress from the labelled training samples is called decision tree induction. The topmost node in a tree is the root node. Let's consider a bank processing of loan application. First at the root node the age group of the applicant is

checked and for age group such as youth and middle age employment status is checked. For old age past history of pending loans checked. According to the answers of such test questions each application is classified to label “safe” or “risk”



The decision tree classifiers are so popular because of the generation of decision tree classifiers is fairly simple and does not require any domain knowledge and specific parameter settings. Decisions trees can easily handle high dimensional data and there by mostly useful in all forms of applications such as bio-medical, financial analysis, manufacturing, production and astronomy and physics. In fact, decisions trees are the basics of several rule-based induction systems such as online shopping sites.

### Feature Selection Measures

Deciding which all features to be considered as the root node at each level of the decision tress a challenging task in a decision tree algorithm. Various methods have been introduced to assign numerical values to the various features such that the values resemble the relative importance of the features. Such measures are called as feature selection measures. There are two widely used feature selection measures, they are information gain and Gini index. In order to understand these features selection methods, one must be familiar with entropy. Let's discuss these terminologies in below section.

## Entropy

The degree to which a subset of examples contains only a single class is known as purity, and any subset composed of only a single class is called a pure class. Simply entropy is a measure of impurity in a dataset. Sets with high entropy are very diverse and provide little information about other items that may also belong in the set, as there is no apparent commonality. The measure of entropy is bits. Entropy values ranges from 0 to 1 when there is only two classes. If there are  $n$  classes, entropy ranges from 0 to  $\log_2(n)$ .

If entropy is minimum it indicates that the sample is homogeneous means less dissimilarity. Whereas the maximum value of entropy indicates that the data is much more diverse

Definition: If  $S$  is a data set with  $c$  class and consider  $p_i$  is the proportion of examples in  $S$  having  $i^{\text{th}}$  class label. Then the entropy of  $S$  is defined as

$$Entropy(S) = \sum_{i=1}^c -p_i \log(p_i)$$

## Information Gain

Consider  $S$  as a set of examples and  $F$  as a feature, also  $S_v$  be the subset of  $S$  with  $F = v$  and all the possible values of feature  $F$  is represented as  $Values(F)$ .

Then the information gain of the feature  $F$  in relation to set  $S$  is given by

$$Gain(S, F) = Entropy(S) - \sum \frac{|S_v|}{|S|} * Entropy(S_v)$$

Here  $|S|$  denotes the total number of elements in  $S$

## Gini Indices

Gini split index of a data set is another feature selection measure used for constriction classification tress.

## Gini index

The Gini index of the data set  $S$ , having  $n$  class labels  $(c_1, c_2, \dots, c_n)$  is defined as

$$Gini(S) = 1 - \sum_{i=1}^r p_i^2$$

Where  $p_i$  represents the proportion of examples having the class label  $c_i$

## Gini split index

Consider  $S$  as a set of examples and  $F$  as a feature, also  $S_v$  be the subset of  $S$  with  $F = v$  and all the possible values of feature  $F$  is represented as  $Values(F)$ .

Then the Gini split index of the feature  $F$  in relation to set  $S$  is given by

$$Ginisplit(S, A) = \sum \frac{|S_v|}{|S|} * Gini(S_v)$$

Here  $|S|$  denotes the total number of elements in  $S$

The following table shows the chance of landslide with respect to the features (rainfall, slope, land use, soil), Let us try to create a decision tree for this table.

S.NO	Rainfall	Slope	Landuse	Soil	Landslide
1	1200-1400	15-30%	Forest	Brown Clay Loam	Low
2	1200-1400	>60%	Agriculture	Brown Clay Loam	High
3	1200-1400	30-60%	Forest	Brown Clay Loam	High
4	1400-1600	15-30%	Forest	Brown Clay Loam	Low
5	1400-1600	>60%	Agriculture	Brown Clay Loam	Low
6	1200-1400	15-30%	Agriculture	Black Clay Soil	High
7	1600-2000	30-60%	Agriculture	Brown Clay Loam	Low

8	1600-2000	30-60%	Forest	Black Clay Soil	High
9	1600-2000	15-30%	Agriculture	Brown Clay Loam	Low

The table having 2 classes,  $C1 = Low$  and  $C2 = High$ . Then probabilities are:

$$P(C1) = \text{number of samples having low class} / \text{Total samples} = 5/9$$

$$P(C2) = \text{number of samples having high class} / \text{Total samples} = 4/9$$

Now let us find the entropy

$$info(D) = \sum_{i=1}^c -p_i \log(p_i) = -5/9 * \log(5/9) - 4/9 * \log(4/9) = 0.991$$

Now find gain for all attributes

a. Rainfall

$$info_{rainfall}(D) = \sum_{j=1}^v \frac{|D_j|}{|D|} * I(D_j)$$

There are three categories of rainfall that is 1200 – 1400, 1400 – 1600, 1600 – 2000

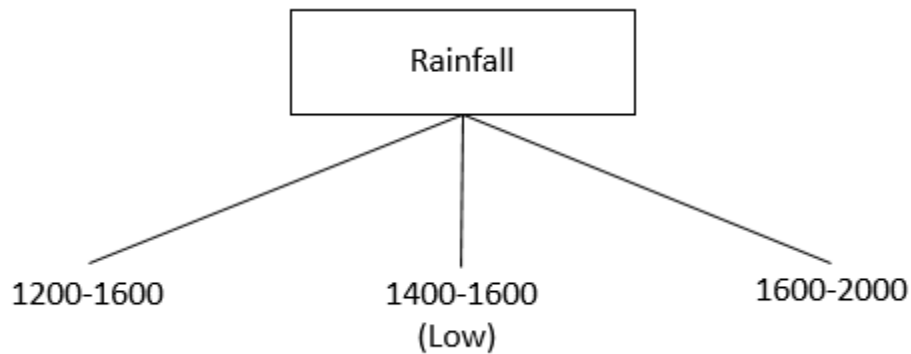
$$\begin{aligned}
 info_{rainfall}(D) &= \frac{4}{9} I(3,1) + \frac{2}{9} I(2,0) + \frac{3}{9} I(2,1) \\
 &= \frac{4}{9} \left( -\left(\frac{1}{4}\right) * \log\left(\frac{1}{4}\right) - \left(\frac{4}{3}\right) \log\left(\frac{4}{3}\right) \right) + \frac{2}{9} * 0 + \frac{3}{9} \left( -\left(\frac{1}{3}\right) * \log\left(\frac{1}{3}\right) - \left(\frac{2}{3}\right) * \log\left(\frac{2}{3}\right) \right) \\
 &= \frac{4}{9} (0.8113) + \frac{3}{9} (0.9183) = 0.6667
 \end{aligned}$$

$$\text{Then } Gain_{rainfall}(D) = info(D) - Info_{rainfall}(D) = 0.3244$$

Similarly you can find the below as

$$Gain_{slope} = 0.1022, Gain_{Landuse} = 0.0072, Gain_{soil} = 0.3197$$

Since rainfall is having the highest gain, we can make it as the root node



For the range 1400-1600 all labels are low

Continuously splitting for other parts

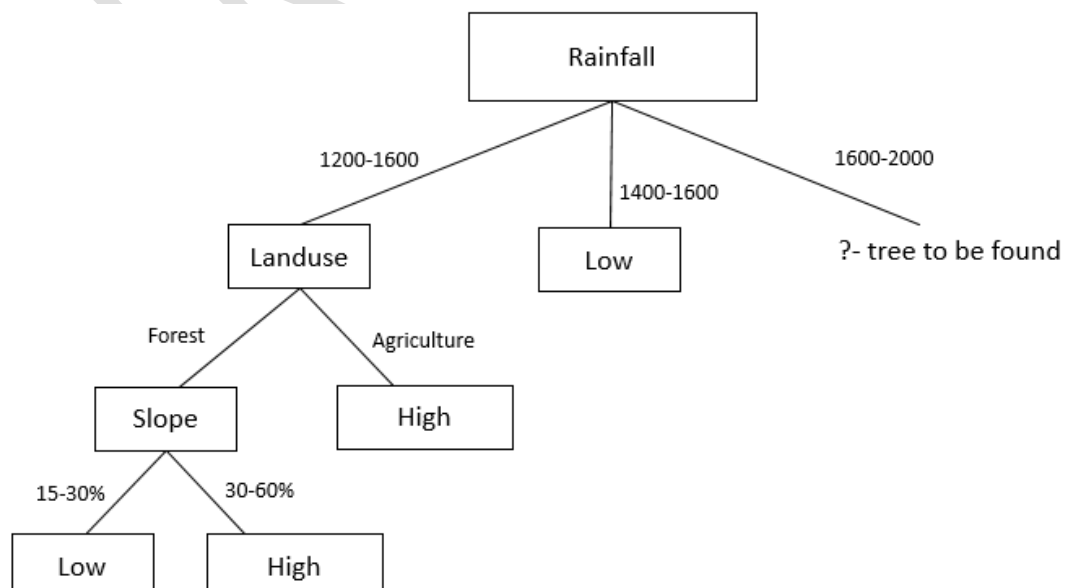
1200-1400 rainfall range, 4 entries for this range

$D = \{1,2,3,4\}$

Using previous equation calculate the gain

$$Gain_{slope} = 0.311278, Gain_{Landuse} = 0.311278, Gain_{soil} = 0.122556$$

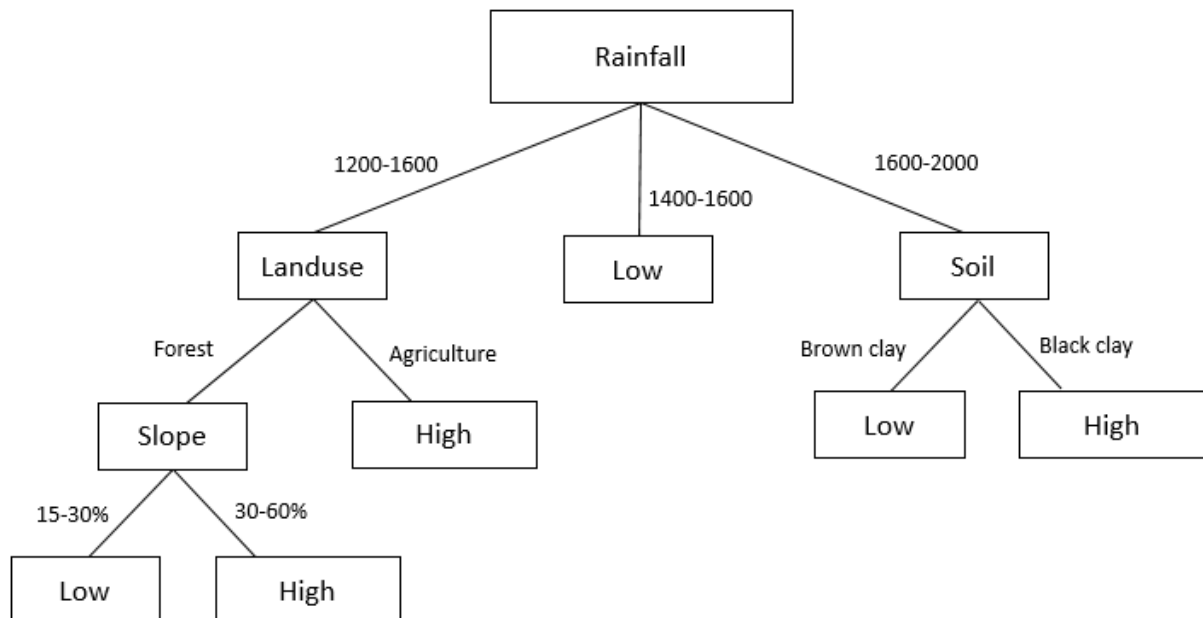
Since  $gain_{slope} = Gain_{Landuse}$  we can select any attribute for splitting, Here we are picking *Landuse* then the tree is given as



Similarly, for the 1600-2000 range, we can find

$$Gain_{slope} = 0.274017, Gain_{Landuse} = 0.918296, Gain_{soil} = 0.918296$$

Let us choose  $Gain_{soil}$  since it is having highest gain, Then the final tree is given as



### **Python Program**

Below given is a python program for classification of iris flowers from iris dataset, a corresponding decision tree is created-classification make use of features sepal length, sepal width, petal length and petal width

```
# import required libraries(DecisionTreeClassifier, export_text) and
dataset(load_iris)
```

```
from sklearn.datasets import load_iris
```

```
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.tree import export_text
```

```
# dataset is loaded to iris variable
```



```

iris = load_iris()

# DecisionTreeClassifier function is called and pass the maximum depth of tree as
parameter

decision_tree = DecisionTreeClassifier(random_state=0, max_depth=5)

# fitting for the data points is done (iris.data is features and iris.target is
datapoints)

decision_tree = decision_tree.fit(iris.data, iris.target)

# tree is generated and saved to r by labelling feature names

r = export_text(decision_tree, feature_names=iris['feature_names'])

# displaying the tree in text form

print(r)

```

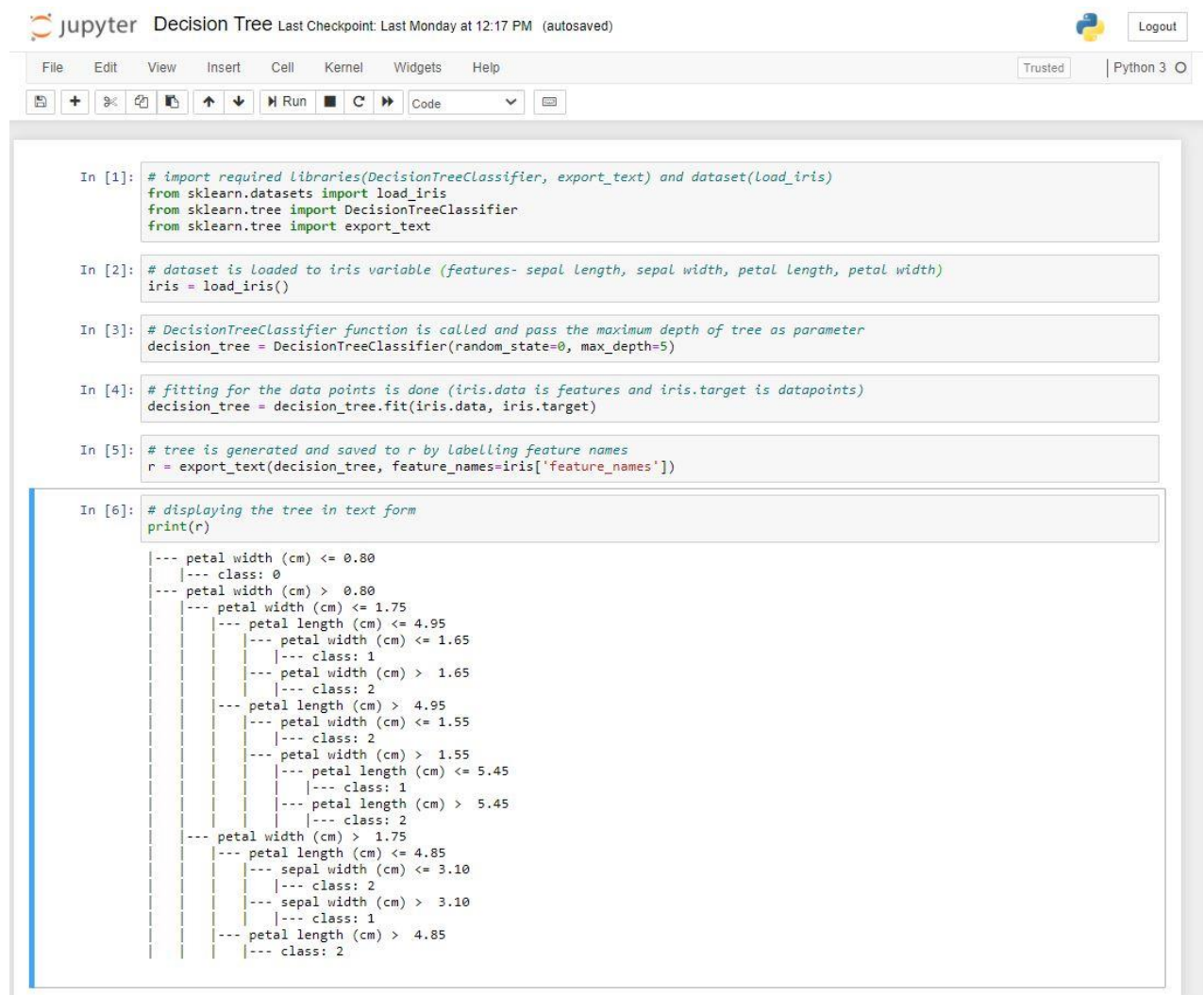
### **Output:**

```

|--- petal width (cm) <= 0.80
|   |--- class: 0
|--- petal width (cm) > 0.80
|   |--- petal width (cm) <= 1.75
|   |   |--- petal length (cm) <= 4.95
|   |   |   |--- petal width (cm) <= 1.65
|   |   |   |   |--- class: 1
|   |   |   |   |--- petal width (cm) > 1.65
|   |   |   |   |   |--- class: 2
|   |   |   |   |--- petal length (cm) > 4.95
|   |   |   |   |   |--- petal width (cm) <= 1.55
|   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |   |--- petal width (cm) > 1.55
|   |   |   |   |   |   |   |--- petal length (cm) <= 5.45
|   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |   |--- petal length (cm) > 5.45
|   |   |   |   |   |   |   |   |   |--- class: 2
|   |   |--- petal width (cm) > 1.75
|   |   |   |--- petal length (cm) <= 4.85
|   |   |   |   |--- sepal width (cm) <= 3.10
|   |   |   |   |   |--- class: 2
|   |   |   |   |   |--- sepal width (cm) > 3.10
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- petal length (cm) > 4.85
|   |   |   |   |   |   |--- class: 2

```

## Jupyter Notebook Screenshot



```
In [1]: # import required libraries(DecisionTreeClassifier, export_text) and dataset(load_iris)
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_text

In [2]: # dataset is loaded to iris variable (features- sepal length, sepal width, petal length, petal width)
iris = load_iris()

In [3]: # DecisionTreeClassifier function is called and pass the maximum depth of tree as parameter
decision_tree = DecisionTreeClassifier(random_state=0, max_depth=5)

In [4]: # fitting for the data points is done (iris.data is features and iris.target is datapoints)
decision_tree = decision_tree.fit(iris.data, iris.target)

In [5]: # tree is generated and saved to r by labelling feature names
r = export_text(decision_tree, feature_names=iris['feature_names'])

In [6]: # displaying the tree in text form
print(r)

|--- petal width (cm) <= 0.80
|--- class: 0
|--- petal width (cm) > 0.80
|--- petal width (cm) <= 1.75
|--- petal length (cm) <= 4.95
|--- petal width (cm) <= 1.65
|--- class: 1
|--- petal width (cm) > 1.65
|--- class: 2
|--- petal length (cm) > 4.95
|--- petal width (cm) <= 1.55
|--- class: 2
|--- petal width (cm) > 1.55
|--- petal length (cm) <= 5.45
|--- class: 1
|--- petal length (cm) > 5.45
|--- class: 2
|--- petal width (cm) > 1.75
|--- petal length (cm) <= 4.85
|--- sepal width (cm) <= 3.10
|--- class: 2
|--- sepal width (cm) > 3.10
|--- class: 1
|--- petal length (cm) > 4.85
|--- class: 2
```

## 2.3. BAYESIAN CLASSIFICATION

Bayesian classifier is a very basic and simple classifier introduced by the understanding of Bayes theorem. This algorithm is used for classifying multiclass datasets. Bayesian classifier often known as naïve Bayes classifier. Before proceeding to Bayes theorem let's understand conditional probability.

### 2.3.1. Conditional probability

Conditional probability can be defined as the probability of occurrence of an event with some relationship to occurrence of one or more other events.

The conditional probability of an event  $A$  given  $B$  denoted as  $P(A|B)$ , ie. The probability of occurrence of an event  $A$  given the probability of occurrence of event  $B$

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \quad \text{if } P(B) \neq 0$$

$P(A \cap B)$  denotes probability of both  $A$  and  $B$  occurring

### 2.3.2. Independent events

Two events are said to be independent if the occurrence of one event does not affect the probability of another event. That is two events  $A$  and  $B$  are said to be independent if  $P(A \cap B) = P(A)P(B)$

Similarly, Three events  $A, B, C$  are said to be mutually independent if

$$P(A \cap B) = P(A)P(B)$$

$$P(B \cap C) = P(B)P(C)$$

$$P(C \cap A) = P(C)P(A)$$

$$P(A \cap B \cap C) = P(A)P(B)P(C)$$

### 2.3.3. Bayes Theorem

Consider two random event  $A$  and  $B$  where  $P(A) \neq 0$ , then the probability of occurrence of  $A$  given  $B$  is

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

$A$  is called proposition and  $B$  is called evidence.  $P(A)$  is the prior probability of proposition and  $P(B)$  is the prior probability of evidence.  $P(A|B)$  is the posterior probability of event  $A$  occurring given event  $B$ .  $P(B|A)$  is the likelihood of event  $B$  occurring given  $A$ .  $P(A)$  and  $P(B)$  are also called as marginal probabilities.

That is

$$\text{likelihood} = \frac{\text{posterior probability} * \text{prior probability}}{\text{marginal probability}}$$

You may generalize Bayes theorem as below. Consider if you have a number of disjoint events  $B_1, B_2, B_3, \dots, B_n$ , in the sample space  $S$  and  $A$  be any event then we have

$$P(B_k|A) = \frac{P(A|B_k)P(B_k)}{\sum_{i=1}^n P(A|B_i)P(B_i)}$$

#### 2.3.4. Naive Bayes algorithm

Naive Bayes algorithm is a simplified approach for classification. Naïve Bayes algorithm operates on the below assumptions.

- All the features considered are independent and does not related to each other, and the presence or absence of any feature does not affect the existence of other feature
- The data under consideration exhibit class-conditional independence, that is the events are independent so long as they are conditioned on the same class value.

Introduction of these assumptions are found to be true for many real-world scenarios and thereby called naïve Bayes algorithm. Naïve means simplified in general context.

Let's move on to the algorithm and idea of Naive Bayes Classifier

Consider a training data set with  $N$  example having  $n$  features. Assume that the features are named as  $(F_1, F_2, \dots, F_n)$  and the feature vector formed as  $(f_1, f_2, \dots, f_n)$ , since it is a supervised technique there is a specific class label associated with each example namely  $(c_1, c_2, \dots, c_t)$  where  $t$  represent the total number of classes.

Now consider we are given with a test instance  $X$  with  $n$  features, that is

$$X = (x_1, x_2, \dots, x_n)$$

Now we have to find and assign most appropriate class label from  $C$  to this test instance  $X$ . In order to assign  $X$  with most appropriate class label we have to find probability of each class labels given test sample  $X$ . That is we have to find

$$P(c_1|X), P(c_2|X), P(c_3|X), \dots, P(c_t|X)$$

After finding each probability, find maximum among them and assign that class label for test instance  $X$ . Suppose  $P(c_j|X)$  gave maximum value then assign class label  $c_j$  for instance  $X$ .

Calculation of probabilities:

As per Bayes theorem  $P(c_k|X)$  is given as

$$P(c_k|X) = \frac{P(X|c_k)P(c_k)}{P(X)}$$

Where  $X$  is set of examples with features  $(x_1, x_2, \dots, x_n)$ . Since we have assumed that there exists a class conditional independence between each of the features in  $X$ , we have the events  $x_1|c_k, x_2|c_k, \dots, x_n|c_k$  all are independent.

Hence, we have

$$P(X|c_k) = P((x_1, x_2, \dots, x_n)|c_k) = P(x_1|c_k)P(x_2|c_k) \dots P(x_n|c_k)$$

Use this in above equation and get

$$P(c_k|X) = \frac{P(x_1|c_k)P(x_2|c_k) \dots P(x_n|c_k)P(c_k)}{P(X)}$$

Since the denominator  $P(X)$  is independent of class labels, we have

$$P(c_k|X) = P(x_1|c_k)P(x_2|c_k) \dots P(x_n|c_k)P(c_k)$$

So, it is enough to find the maximum among the following values:

$$P(x_1|c_k)P(x_2|c_k) \dots P(x_n|c_k)P(c_k), k = 1, 2, \dots, t$$

$$P(c_k) = \frac{\text{number of example with class label } c_k}{\text{total number of examples}}$$

$$P(x_j|c_k) = \frac{\text{no of examples with } j\text{th feature equal to } x_j \text{ and class label } c_k}{\text{no of example with class label } c_k}$$

### Algorithm

Consider a training data set with  $N$  example having  $n$  features. Assume that the features are named as  $(F_1, F_2, \dots, F_n)$  and the feature vector formed as  $(f_1, f_2, \dots, f_n)$ , since it is a supervised technique there is a specific class label associated with each example namely  $(c_1, c_2, \dots, c_t)$  where  $t$  represent the total number of classes.

Now consider we are given with a test instance  $X$  with  $n$  features, that is

$$X = (x_1, x_2, \dots, x_n)$$

Step 1: Compute  $P(c_k)$  for  $k = 1, 2, 3, \dots, t$

Step 2: Generate a table of conditional probabilities.

$$P(f_1|c_k), P(f_2|c_k), \dots, P(f_n|c_k) \text{ for all } k = 1, 2, \dots, t$$

Step 3: Calculate product  $p_k = P(x_1|c_k)P(x_2|c_k) \dots P(x_n|c_k)P(c_k)$ , for each  $k = 1, 2, \dots, t$

Step 4: Find a label  $c_j$  such that  $p_j = \max(p_1, p_2, \dots, p_t)$

Step 5: Assign  $c_j$  to test instance  $X$

Let us examine the working of the algorithm based on an example. Consider the following data having frequency of fruits with 3 features (Yellow, Sweet, Long) and

classify a new sample to either mango or Banana or other fruit. The new fruit is Yellow, Sweet and Long

Fruit	Yellow	Sweet	Long	Total
Mango	350	450	0	650
Banana	400	300	350	400
Other	50	100	50	150
Total	800	850	400	1200

$X = \{Yellow, Sweet, Long\}$

1.  $X$  be a mango

$$P(X|mango) = P(Yellow|Mango) * P(Sweet|Mango) * P(Long|Mango)$$

$$P(Yellow|Mango) = \frac{P(Mango|Yellow)*P(Yellow)}{P(Mango)} = \frac{\frac{350}{800} * \frac{800}{1200}}{\frac{650}{1200}} = 0.53$$

$$P(Sweet|Mango) = \frac{P(Mango|Sweet)*P(Sweet)}{P(Mango)} = \frac{\frac{450}{850} * \frac{850}{1200}}{\frac{650}{1200}} = 0.69$$

$$P(Long|Mango) = \frac{P(Mango|Long)*P(Long)}{P(Mango)} = \frac{\frac{0}{400} * \frac{400}{1200}}{\frac{650}{1200}} = 0$$

$$P(X|mango) = 0.53 * 0.69 * 0 = 0$$

2.  $X$  be a Banana

$$P(X|Banana) = P(Yellow|Banana) * P(Sweet|Banana) * P(Long|Banana)$$

$$P(Yellow|Banana) = \frac{P(Banana|Yellow)*P(Yellow)}{P(Banana)} = \frac{\frac{400}{800} * \frac{800}{1200}}{\frac{400}{1200}} = 1$$

$$P(Sweet|Banana) = \frac{P(Banana|Sweet)*P(Sweet)}{P(Banana)} = \frac{\frac{300}{850} * \frac{850}{1200}}{\frac{400}{1200}} = 0.75$$

$$P(\text{Long}|\text{Banana}) = \frac{P(\text{Banana}|\text{Long}) * P(\text{Long})}{P(\text{Banana})} = \frac{\frac{350}{400} * \frac{400}{1200}}{\frac{400}{1200}} = 0.875$$

$$P(X|\text{Banana}) = 1 * 0.75 * 0.875 = 0.65$$

3.  $X$  be other fruit

$$P(X|\text{Others}) = P(\text{Yellow}|\text{Others}) * P(\text{Sweet}|\text{Others}) * P(\text{Long}|\text{Others})$$

$$P(\text{Yellow}|\text{Others}) = \frac{P(\text{Others}|\text{Yellow}) * P(\text{Yellow})}{P(\text{Others})} = \frac{\frac{50}{800} * \frac{800}{1200}}{\frac{150}{1200}} = 0.33$$

$$P(\text{Sweet}|\text{Others}) = \frac{P(\text{Others}|\text{Sweet}) * P(\text{Sweet})}{P(\text{Others})} = \frac{\frac{100}{850} * \frac{850}{1200}}{\frac{150}{1200}} = 0.66$$

$$P(\text{Long}|\text{Others}) = \frac{P(\text{Others}|\text{Long}) * P(\text{Long})}{P(\text{Others})} = \frac{\frac{50}{400} * \frac{400}{1200}}{\frac{150}{1200}} = 0.33$$

$$P(X|\text{Others}) = 0.33 * 0.66 * 0.33 = 0.071$$

Since  $P(X|\text{Banana}) > P(X|\text{Mango}) > P(X|\text{Others})$ ,  $X$  is classified as Banana

## 2.4. SUPPORT VECTOR MACHINE (SVM)

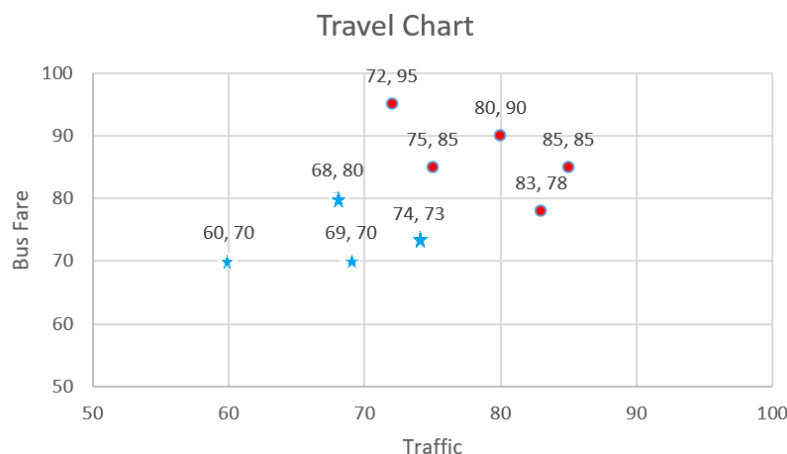
Support vector machines is a form of supervised learning technique to solve classification problems, and this technique aims to find a best hyperplane which classify data set to different classes. Where the support vectors are the data points closest to the created hyperplane, these are very critical since these points lies close to the hyperplane hence the removal of these points could alter the position of hyperplane.

We will examine Support Vector Machines through an example scenario. Consider the data for determining the chances of travel, for simplicity let's consider there are only two features we are looking at to determine the travel decision such as traffic and bus fare. The output will be either yes or no, that is if yes travel is possible, and if no travel is not done.

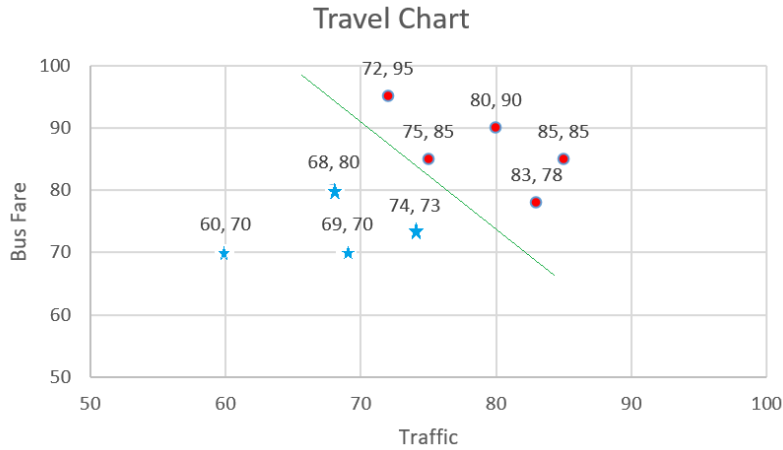


Traffic	Bus fare	Travel possibility
85	85	No
60	70	Yes
80	90	No
72	95	No
68	80	Yes
74	73	Yes
69	70	Yes
75	85	No
83	78	No

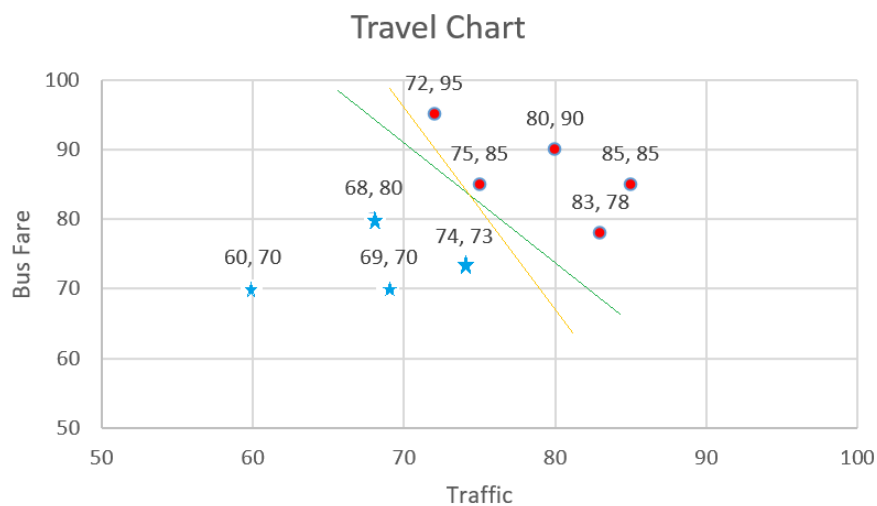
Since the output is either yes or no, this can be considered as a two class problem. And the dataset is said to be two class dataset. Now let's consider the scatter plot of the above data. The above data can be plotted on a 2-D plane keeping features on the axis of the plot. Here we will be generating the scatter plot keeping traffic in  $x$  axis and bus fare in  $y$  axis. Here in this plot blue star represent class label *yes* and red circle denotes class label *no*



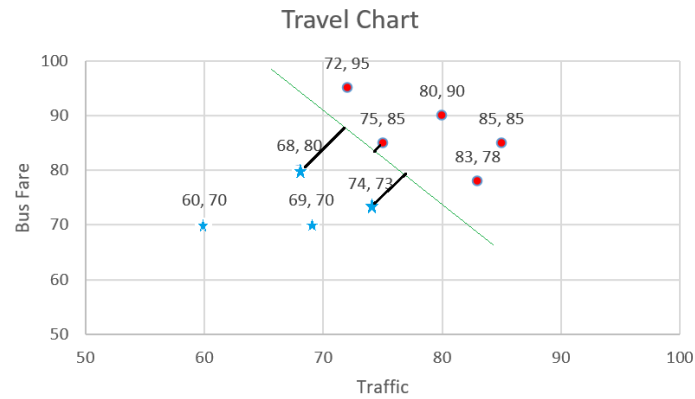
As we examine the above diagram, we found that it was possible to draw a straight line in the scatter diagram to separate the values which produced an output *yes* and *no*. That is given below, the green line separate in such a way that all the points marked with blue star is in one side of the line and the points marked with red circles are on the other side of the line. Such line is called a separating line. If there exist such a line for a given dataset, the data is said to be linearly separable, but there are other data that are not linearly separable. Data in our example scenario is linearly separable.



In our above example, one could draw multiple line to separate these classes, below shows another yellow line drawn to separate the classes *yes* and *no*. In this case selection of best separator line is important.

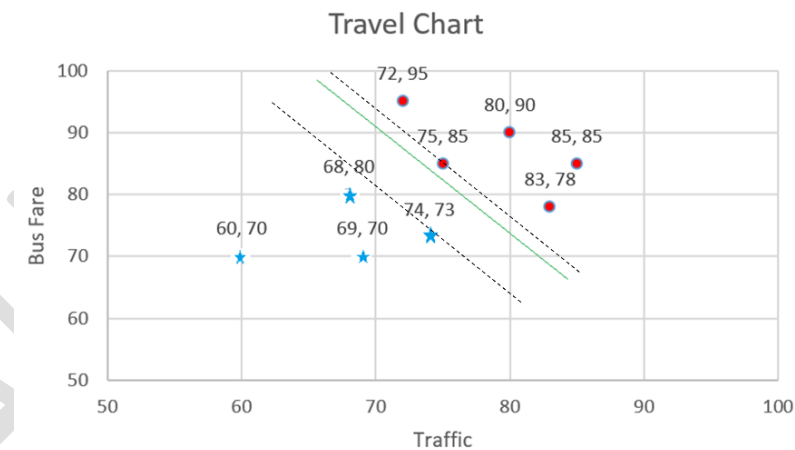


The concept of the margin of a separating line can be used to find the best separator line. Consider the perpendicular distance between the data points and the separator line. The margin of the separator line is defined as the double of the shortest perpendicular distance between data points and separator line. Below figure shows some of the perpendicular distance and least perpendicular distance for our data. The separator line with maximum margin can be considered as the best separator line and choose for classification.



In the above figure, you can find the shortest perpendicular distance from the line is the one to the data point (75,85). The line with highest value for margin is called “maximum margin line” or the “optimal separating line”. This line is otherwise called as “support vector machine”, where “support vector” is the data points that are closest to the optimal separating line.

We may draw a dashed line through the nearest support vector of both classes as shown in figure.



The region between these two dashed lines can be thought of as a “road” or a “street” of maximum width that separates the “yes” data points and the “no” data points.

## **Python Program**

```
# import numpy, StandardScaler, SVC- standard library packages to implement SVM
based classification

import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

# input X and y- X being the data points and y being the class labels.

X = np.array([[-1, -1], [-2, -1], [1, 1], [2, 1]])
y = np.array([1, 1, 2, 2])

# Perform the SVM classification using SVC()function by setting gamma parameter to
'auto'

clf=SVC(gamma='auto')
clf.fit(X, y)

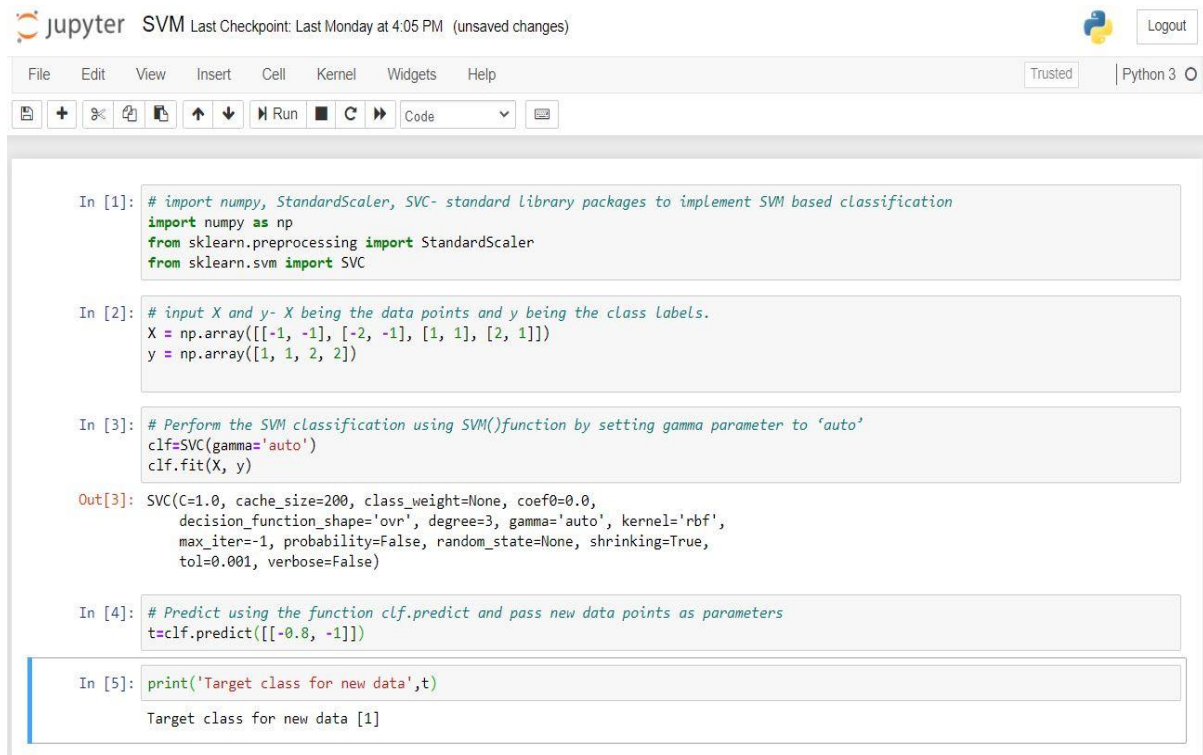
# Predict using the function clf.predict and pass new data points as parameters

print(clf.predict([[-0.8, -1]]))
```

### **Output:**

**[1]**

## Jupyter Notebook Screenshot



The screenshot shows a Jupyter Notebook titled 'SVM' with a last checkpoint from 'Last Monday at 4:05 PM' and '(unsaved changes)'. The interface includes a top bar with 'jupyter' logo, a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), and a status bar (Trusted, Python 3). The notebook contains five input cells and one output cell. The code in the input cells performs the following steps: 1. Imports numpy, StandardScaler, and SVC from sklearn. 2. Defines data points X and class labels y. 3. Fits an SVC model with gamma='auto'. 4. Predicts the class for new data points. 5. Prints the target class for new data. The output cell shows the result of the prediction: 'Target class for new data [1]'.

```
In [1]: # import numpy, StandardScaler, SVC- standard library packages to implement SVM based classification
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

In [2]: # input X and y- X being the data points and y being the class labels.
X = np.array([[-1, -1], [-2, -1], [1, 1], [2, 1]])
y = np.array([1, 1, 2, 2])

In [3]: # Perform the SVM classification using SVM() function by setting gamma parameter to 'auto'
clf=SVC(gamma='auto')
clf.fit(X, y)

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

In [4]: # Predict using the function clf.predict and pass new data points as parameters
t=clf.predict([[-0.8, -1]])

In [5]: print('Target class for new data',t)

Target class for new data [1]
```

### 2.4.1. Hyper plane

Hyper planes are said to be the subset of finite dimensional spaces, which are similar to straight lines and planes in three-dimensional space. It can be considered as a plane whose dimension one less than that of vector space. That is if you consider a vector space of dimension 3 then the corresponding hyper plane have a dimension of 2.

If you have two features, then the resulting vector space is of 2 dimension and is separated by a hyperplane of dimension 1 which is a line.

### 2.4.2. Two class dataset

In machine learning a two-class dataset is said to be a dataset in which the target variable takes only one of the two possible class labels. The variable whose value is being predicted is called target variable or output variable, in our earlier example the

*travel possibility* was target variable. If the target variable takes more than two values then such dataset is said to be a multi class data set. Our earlier dataset to decide travel possibility, has only two values for the target variables either *yes* or *no*.

Let us consider a two-class dataset having  $n$  number of features and let the class label be  $+1$  and  $-1$ . Let the vector  $x = (x_1, x_2, \dots, x_n)$  represent the values of the features for one instance. This data is said to be linearly separable if we can find a hyperplane having dimension  $n - 1$ . Say the hyperplane as given below

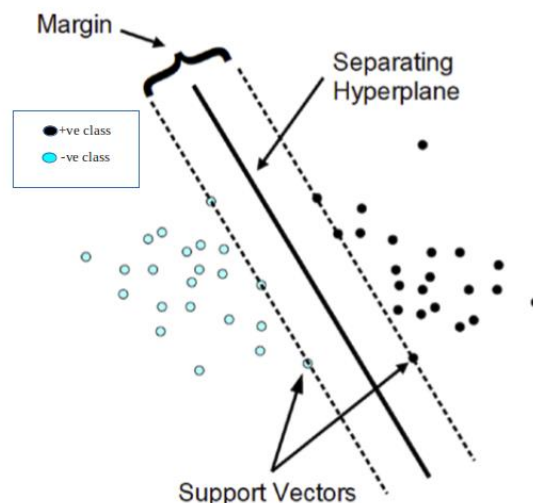
$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n = 0$$

Having following properties.

For each instance in  $x$  with class label  $+1$ ;  $\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n > 0$

For each instance in  $x$  with class label  $-1$ ;  $\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n < 0$

### 2.4.3. Maximal margin hyperplanes



- Consider a hyperplane  $H$  for a linear separable dataset having target variable values either  $+1$  or  $-1$
- Consider the perpendicular distance from the separating hyperplane  $H$  to the training instances  $x$ .

- The double of the smallest perpendicular distance is called margin of the hyperplane  $H$ .
- The hyperplane having maximum margin is called maximal margin hyperplane or optimal hyperplane.
- The support vector machine for the dataset is said to be the maximal margin hyperplane and the data points that lies closest to the hyperplane is called support vectors.

## 2.5. K-NEAREST NEIGHBOUR CLASSIFICATION

The k nearest neighbour classification works by comparing a given test example with training examples, since the training examples output class labels are given this is a supervised learning technique. Although the technique was introduced in the early 50 it didn't not used widely because of lack of computational power. The technique become popular as the availability of computational power increased. As we discussed in earlier classification techniques, here also the training set consist of  $n$  features. An instance of training example can be considered as  $x = (x_1, x_2, \dots, x_n)$

K nearest neighbour classify by finding  $k$  training examples that are closest to the unknown example. These  $k$  training tuples are the  $k$  "nearest neighbours" of the unknown example. Now one has to understand how the closeness is defined, you can consider closeness in terms of a distance metric such as Euclidean distance. Euclidean distance between two examples  $X_1 = (x_{11}, x_{12}, \dots, x_{1n})$  and  $X_2 = ((x_{21}, x_{22}, \dots, x_{2n}))$  is given as

$$distance(X_1, X_2) = \sqrt{\sum_{i=1}^n (x_{1i} - x_{2i})^2}$$

In simple terms, for each numeric attributes, we take the difference between the corresponding values of attribute in example  $X_1$  and  $X_2$ . Then a square root is computed for the sum of squares of these differences. Since the K-nearest neighbour classification technique consists of taking differences, squares and square roots, we apply normalization on features. This normalization of features helps to use smaller values instead of larger numeric values. For example, a feature like *profit* often has

large values, normalizing this feature can reduce the computational complexity and memory requirement. Min-max normalization is a type of normalization that can be used to normalize a feature and make their values fall in a range of [0,1]

Consider  $v$  as a value of a numeric feature  $A$  of an example  $X$ , and the normalized value  $v'$  for the feature can be found as

$$v' = \frac{v - \min_A}{\max_A - \min_A}$$

Where  $\min_A$  and  $\max_A$  refers to the minimum and maximum value of the attribute  $A$

K-nearest-neighbour classification assign the most common class among its  $k$  nearest neighbours for the unknown example. That is when  $k = 1$ , the unknown tuple is assigned the class of the training tuple that is closest to it in pattern space. The distance measure is possible for numerical features, now consider a feature having categorical values such as "colour". For understanding this consider a feature colour having value *blue* in example  $X_1$  and value *red* in example  $X_2$ . Then the difference between the two can be considered as 1, and if values for the feature is *blue* for both examples, then the difference can be considered as 0.

Missing values also need to be taken in to consideration before proceeding to this classification method. If the numerical value of a feature is missing from either one of the example, then we can assume maximum possible difference. Another problem to be addressed is the decision a better  $k$  value. We decide this by experimentally, we may start with  $k = 1$ , then we use a test set to estimate the error ratio of the classifier, this process can be continued by incrementing  $k$  gradually and finding a  $k$  value which produce a minimum error ratio for the classifier. In general, the larger the number of training examples is, the larger the value of  $k$  will be.

Although we have used Euclidean distance as a distance measure, other distance measures such as Manhattan distance or Mahala Nobis distance. Now let us see the working of K-nearest neighbour classification via an example

Consider the given dataset and find the class label for the sample  $X$  having features (3,7)



Length	Strength	Class
7	7	Bad
7	4	Bad
3	4	Good
1	4	Good

Suppose  $K = 3$

Calculate the distance between the queried sample and set of given samples using Euclidean distance formula

$$D1 = \sqrt{(7 - 3)^2 + (7 - 7)^2} = 4$$

$$D2 = \sqrt{(7 - 3)^2 + (4 - 7)^2} = 5$$

$$D3 = \sqrt{(3 - 3)^2 + (4 - 7)^2} = 3$$

$$D4 = \sqrt{(1 - 3)^2 + (4 - 7)^2} = 3.60$$

The queried samples having 3 nearest neighbours with distance measures 3, 3.6, and 4. They having class labels Good, Good and Bad.

Using the majority of the category, we classified our query sample (3,7) to Good.

### **Python Program**

```
# import numpy and KNeighborsClassifier from sklearn.neighbors
```

```
from sklearn.neighbors import KNeighborsClassifier
```

```
import numpy as np
```

```
# input Data points X and classes y(0-bad,1-good)
```

```
X = [[7, 7], [7, 4], [3, 4],[1, 4]]
```

```
y = [0, 0, 1, 1]
```

```
# Perform k nearest neighbour classification using the function,  
KNeighborsClassifier and give neighbour parameters as 3
```

```
neigh = KNeighborsClassifier(n_neighbors=3)  
neigh.fit(X, y)  
KNeighborsClassifier(...)
```

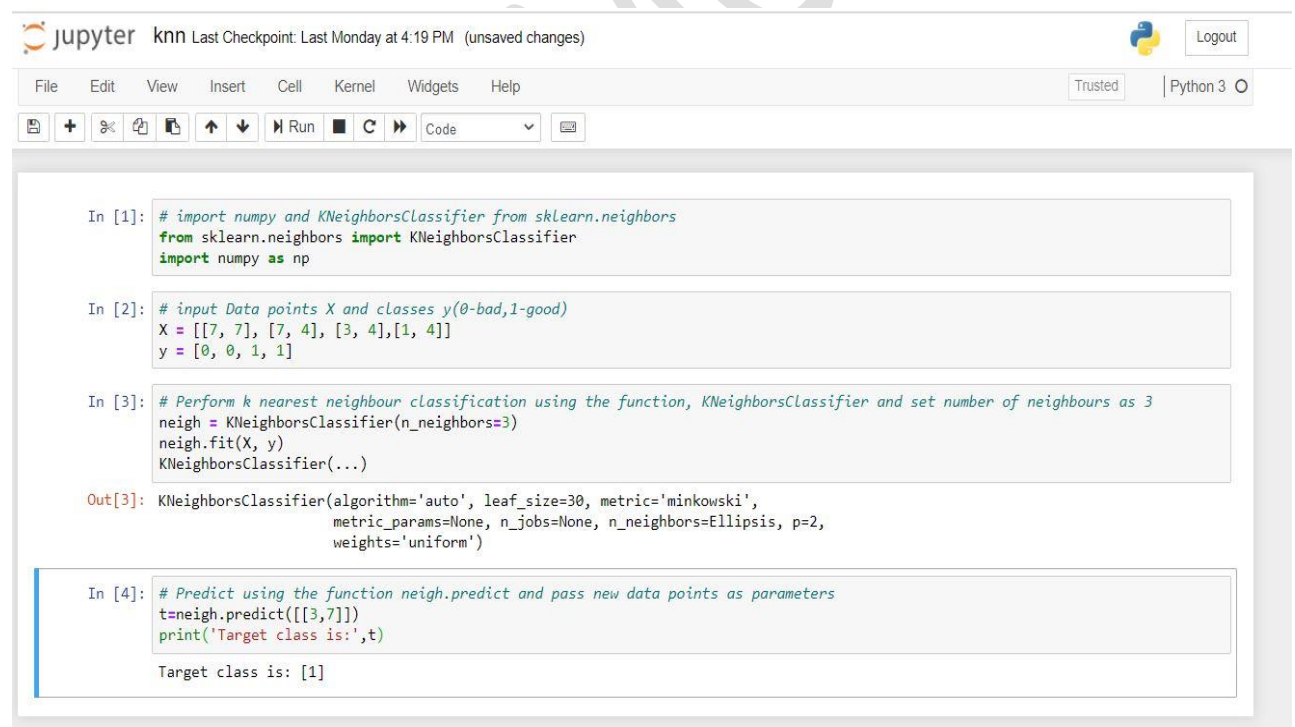
```
# Predict using the function neigh.predict and pass new data points as parameters
```

```
print(neigh.predict([[3,7]]))
```

### **Output:**

```
[1]
```

### **Jupyter Notebook Screenshot**



The screenshot shows a Jupyter Notebook interface with the title 'knn' and a last checkpoint from 'Last Monday at 4:19 PM'. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The notebook contains four input cells and one output cell. The first three input cells contain code for importing libraries, defining data points, and training a KNeighborsClassifier. The fourth input cell contains code for predicting a new data point. The output cell shows the result of the prediction.

```
In [1]: # import numpy and KNeighborsClassifier from sklearn.neighbors  
from sklearn.neighbors import KNeighborsClassifier  
import numpy as np  
  
In [2]: # input Data points X and classes y(0-bad,1-good)  
X = [[7, 7], [7, 4], [3, 4], [1, 4]]  
y = [0, 0, 1, 1]  
  
In [3]: # Perform k nearest neighbour classification using the function, KNeighborsClassifier and set number of neighbours as 3  
neigh = KNeighborsClassifier(n_neighbors=3)  
neigh.fit(X, y)  
KNeighborsClassifier(...)  
  
Out[3]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',  
                             metric_params=None, n_jobs=None, n_neighbors=Ellipsis, p=2,  
                             weights='uniform')  
  
In [4]: # Predict using the function neigh.predict and pass new data points as parameters  
t=neigh.predict([[3,7]])  
print('Target class is:',t)  
  
Target class is: [1]
```

## CHAPTER 3

### UNSUPERVISED MACHINE LEARNING

Unsupervised learning is the type of learning technique in which the learning algorithm tries to label previously undetected data points with very minimal human supervision. This type of learning technique involves the training of machines using the information's that are neither classified or labelled earlier, and the learning process continue without any supervision as compared to supervised learning techniques. The machine tries to classify the data point without any prior information, by identifying the patterns, similarities and differences.

Let's understand the learning situation and algorithm operation using a very simple example, consider the system is given an image having both cars and scooters, and let's say that the machine have neither seen a car or scooter before. In this case our machine does not have any idea of the features of either car or scoter, but the machine is given the learning ability so that it can categorize cars and scooters from the image by identifying their similarities, differences and patterns. Unsupervised learning generally allows machines to work with unlabelled data.

There are two categories of unsupervised learning:

- Clustering: Clustering is a process which divides a set of data points to a number of groups such that the similarity between data points with in same group is high and the similarity between data points in different group is less. These groups are said to be clusters and the similarity between data points within same cluster is said to be intraclass similarity and similarity between data points of different clusters is said to be interclass similarity.
- Association: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

There are different types of clustering techniques and we will discuss each technique in below section

## Clustering Types:-

1. K-means clustering
2. K-NN (k nearest neighbours)
3. K-medoid clustering or PAM (partitioning around medoids)
4. Fuzzy C means clustering
5. DBSCAN

### 3.1. K-MEANS CLUSTERING

K means clustering algorithm is one of the basic and simple unsupervised learning algorithms for solving the clustering problem. Consider that we have to classify the data set into  $k$  clusters. We start the clustering process by selecting  $k$  random points as the initial cluster centres for the  $k$  number of clusters. Now distance measure is computed between the data points and each cluster centre of  $k$  clusters. The data point which is having minimal distance to the cluster centre is grouped to that cluster. After each data points is associated corresponding cluster centres, we take the averages of the data points associated with a centre and replace the centre with the average, and this is done for each of the centres. The process is repeated until the centres converge to some fixed points. After the clusters are created, the data points in each cluster exhibit maximum intra class similarity and minimal interclass similarity.

The distance measure used can be Euclidian distance. Consider two data points  $(x_1, x_2)$  and  $(y_1, y_2)$  then the distance between these points can be calculated as

$$\sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

Let us consider each data point having  $n$  features, then the datapoint is a  $n$  dimensional vector. That is

$$x = (x_1, x_2, \dots, x_n)$$

Let us consider another data point  $y$  as

$$y = (y_1, y_2, \dots, y_n)$$

Then the distance between these two data points is given as

$$\sqrt{(x_1 - y_1)^2 + \dots + (x_n - y_n)^2}$$

Consider a set of data points  $X$  having  $N$  number of datapoints, where each data point is defined as a  $n$  dimensional vector, and let  $V = (v_1, v_2, \dots, v_k)$  be the set of centres and  $c_i$  be the number of data points in the  $i^{\text{th}}$  cluster when  $i = 1, 2, \dots, k$ . Here  $k$  is the total number of desired clusters. Here the basic idea of a clustering algorithm is to partition  $X$  to a  $k$  disjoint subset  $S = (S_1, S_2, \dots, S_k)$

### 3.1.1. K- means clustering algorithm

Step 1: Select  $k$  cluster centres randomly  $V = (v_1, v_2, \dots, v_k)$

Step 2: Calculate distance between each data points and each cluster centre in  $V$

Step 3: Assign each data points  $x_j$  to the cluster center  $v_i$  if the  $\text{dist}(x_j, v_i)$  is Minimum

Step 4: Let's say  $(x_{i1}, x_{i2}, \dots, x_{ic_i})$  be the data points assigned to cluster centre  $v_i$ , where  $c_i$  is the total number of datapoints in class  $c_i$

Step 5: Recalculate the cluster centres by taking mean of the data points in that cluster. Use the below equation

$$v_i = \frac{1}{c_i} (x_{i1} + x_{i2} + \dots + x_{ic_i}), i = 1, 2, 3, \dots, k$$

Step 6: Now recalculate the distance measure for each data points with the newly assigned cluster centres, and stop if there is a convergence for the new cluster centres.

### How to determine initial cluster centres?

The following steps can be taken to initialize cluster centres

- Random selection of  $k$  datapoints from the dataset for  $k$  clusters
- Another method is to take the average of whole data points and add some random numbers with it to generate  $k$  cluster centres
- Divide the data points to  $k$  equal intervals so that the data is partitioned into  $k$  number of groups. Now take mean of each group to assign as  $k$  cluster centres.

Although the K-means clustering algorithm is simple it has certain disadvantages such as

- Random selection of cluster centres may not lead to an optimal result
- The algorithm can work only on numerical data, it can't be applied for categorical data
- The structure of final cluster depends on initial cluster centre selection
- The number of clusters is specified by user before execution of algorithm

Let us illustrate k-means clustering with an example. Consider the following dataset and cluster data into two clusters

Sample	Feature 1	Feature 2
A	1	1
B	2	1
C	4	3
D	5	4

Assume initial centroid of two clusters C1 and C2 are A and B

Calculate the distance between centroid and each point using Euclidean distance

For centroid A

$$D(A, A) = 0$$

$$D(A, B) = \sqrt{(1 - 2)^2 + (1 - 1)^2} = 1$$

$$D(A, C) = \sqrt{(1 - 4)^2 + (1 - 3)^2} = 3.61$$

$$D(A, D) = \sqrt{(1 - 5)^2 + (1 - 4)^2} = 5$$

For centroid  $B$

$$D(B, A) = \sqrt{(2 - 1)^2 + (1 - 1)^2} = 1$$

$$D(B, B) = 0$$

$$D(B, C) = \sqrt{(2 - 4)^2 + (1 - 3)^2} = 2.83$$

$$D(B, D) = \sqrt{(2 - 5)^2 + (1 - 4)^2} = 4.24$$

Assign each Samples based on minimum distance

$$D(A, C) > D(B, C), C \text{ is in cluster } C2$$

$$D(A, D) > D(B, D), D \text{ is in cluster } C2$$

Hence  $C1$  contain -A

$C2$  contains-  $B, C, D$

Determine new centroid

$$C1 = (1, 1)$$

$$C2 = (2 + 4 + 5/3, 1 + 3 + 4/3) = (11/3, 8/3)$$

Repeat the distance calculation and update the clusters using new centroid. Repeat this procedure until no change in centroid.

### Python Program

```
# import required libraries numpy and Kmeans from sklearn.cluster

from sklearn.cluster import KMeans
import numpy as np

# load the data points to X

X = np.array([[1, 1], [2, 1], [4, 3], [5, 4]])
# perform k means clustering for n=2
kmeans = KMeans(n_clusters=2, random_state=0).fit(X)

#printing cluster labels for data points in X
kmeans.labels_

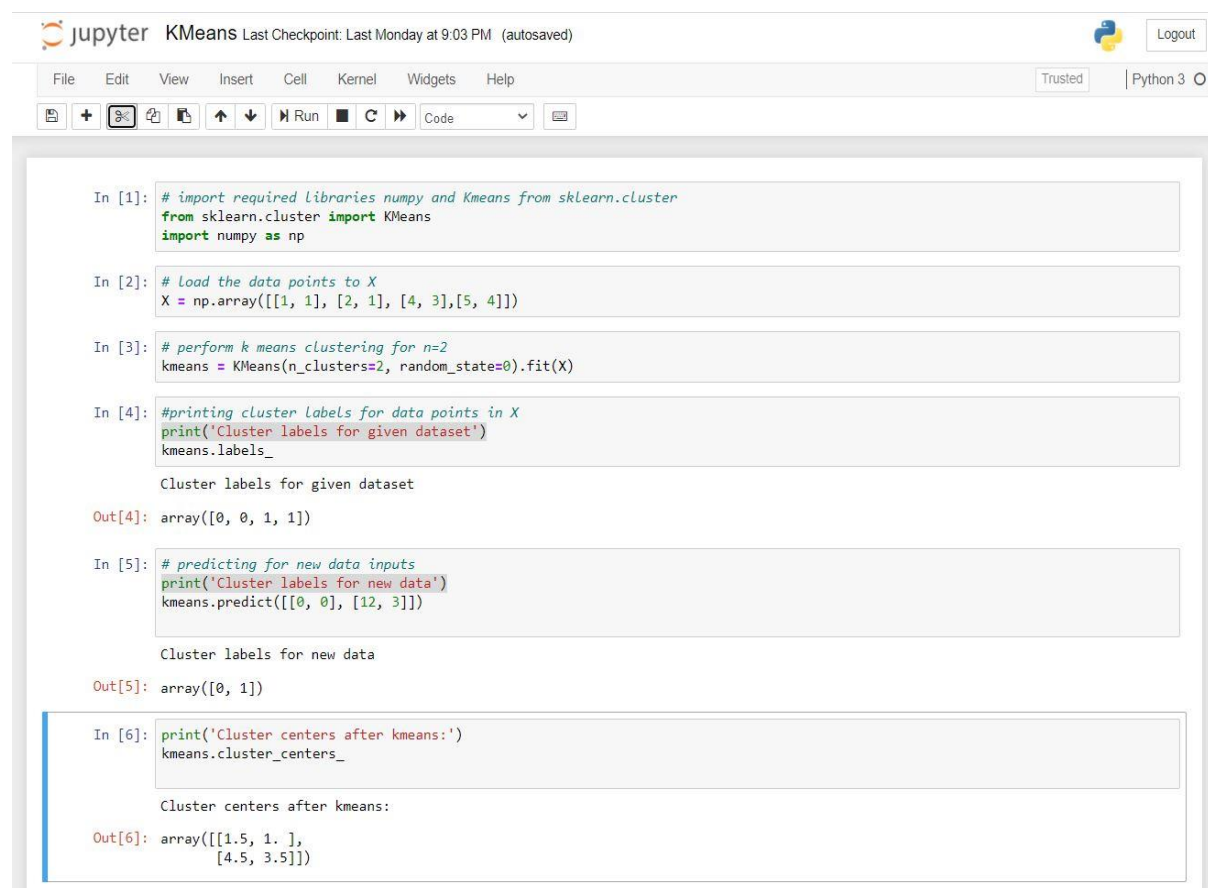
# predicting for new data inputs

kmeans.predict([[0, 0], [12, 3]])
```

**Output:**

```
array([0, 0, 1, 1])  
array([0, 1])
```

## Jupyter Notebook Screenshot



```
jupyter KMeans Last Checkpoint: Last Monday at 9:03 PM (autosaved)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 O

In [1]: # import required libraries numpy and Kmeans from sklearn.cluster
from sklearn.cluster import KMeans
import numpy as np

In [2]: # Load the data points to X
X = np.array([[1, 1], [2, 1], [4, 3], [5, 4]])

In [3]: # perform k means clustering for n=2
kmeans = KMeans(n_clusters=2, random_state=0).fit(X)

In [4]: #printing cluster labels for data points in X
print('Cluster labels for given dataset')
kmeans.labels_

Cluster labels for given dataset

Out[4]: array([0, 0, 1, 1])

In [5]: # predicting for new data inputs
print('Cluster labels for new data')
kmeans.predict([[0, 0], [12, 3]])

Cluster labels for new data

Out[5]: array([0, 1])

In [6]: print('Cluster centers after kmeans:')
kmeans.cluster_centers_

Cluster centers after kmeans:

Out[6]: array([[1.5, 1. ],
               [4.5, 3.5]])
```

## 3.2. K-NN (K NEAREST NEIGHBOURS) CLUSTERING

This algorithm is very simple as k-nearest neighbour classification, In  $k$  nearest neighbour clustering data points are iteratively merged into existing clusters that are nearby. A threshold or a distance measure  $t$  can be used to determine if the incoming data points to be added to existing cluster or to create a new cluster.

### Algorithm for $K - NN$

Step 1: Consider  $X = (x_1, x_2, \dots, x_n)$  be the data points to be clustered.



Step 2: Create a adjacency matrix  $A$  showing distance between data points.

Step 3: Initialize  $K$  as the set of clusters.

Step 4: Add first data point  $x_1$  to the set of clusters  $K$

Step 5: Iteratively find a  $x_i$  where the  $dist(x_1, x_i)$  is the smallest

Step 6: if  $dist(x_1, x_i)$  is less than the predefined distance threshold  $t$  then map  $x_i$  to same cluster else map  $x_i$  to the set of clusters  $K$

Step 7: Repeat the process over all data points to complete clustering

### 3.3. K-MEDOID CLUSTERING OR PAM (PARTITIONING AROUND MEDOIDS)

K medoid clustering is otherwise called as PAM (partitioning around medoid), in this type of clustering, each clusters are represented by a medoid. This method effectively handles outliers since it is using medoids for clustering. Outliers are the data points that could not be included to any of the clusters, it can either be ignored or considered as a new cluster.

We start the K-medoid clustering by taking a random set of  $k$  data points as the set of medoids. Then at each step, all data points that are not medoids are analysed one by one to see if they should be medoids. As we progress, the algorithm decides whether there is an item among data points to replace existing medoids. The algorithm chooses the pair that improves the overall quality of the clustering the best and exchanges them. Sum of all distances from a non-medoid object to the medoid for the cluster it is in is computed to determine the quality of clustering. An item is assigned to the cluster represented by the medoid to which it is closest (minimum distance)

Let us consider that medoid  $t_i$  represents a cluster  $K_i$ , now we take a decision to swap medoid  $t_i$  with another non medoid data point  $t_h$  only if such swapping decision shows an improvement to the overall impact to the cost. Here the cost function is nothing but the sum of distances to cluster medoids. Now let us examine cost function

Consider the  $C_{jih}$  is the cost change for an item  $t_j$  associated with swapping medoids  $t_i$  with non medoid  $t_h$ . The cost is the change to the sum of all distances from items to their cluster medoids. The total impact occur in the quality due to a medoid change is given as

$$TC_{ih} = \sum_{j=1}^n C_{jih}$$

#### Algorithm for $K$ – medoid or PAM

Step 1: Consider  $X = (t_1, t_2, \dots, t_n)$  be the data points to be clustered.

Step 2: Create a adjacency matrix  $A$  showing distance between data points.

Step 3: Initialize  $k$  as the number of desired clusters and consider  $K$  (set of clusters) as the output

Step 4: Select  $k$  medoids randomly from set of data points

Step 5: For each non medoid  $t_h$ , for each medoid  $t_i$  calculate  $TC_{ih}$

Step 6: Find  $i$  and  $h$  where  $TC_{ih}$  is the smallest

Step 7: If  $TC_{ih} < 0$  then replace medoid  $t_i$  with  $t_h$

Step 8: For each  $t_i \in X$ , assign  $t_i$  to  $K_j$ , if distance between  $t_i$  and  $t_j$  is the smallest over all medoids

### **3.4. FUZZY C MEANS CLUSTERING**

Fuzzy clustering is a form of clustering applicable when a data point can exist in more than one cluster. Fuzzy clustering is otherwise called as soft  $k$ -means or soft clustering. Each data point exhibits a probability to belong to each cluster, we can say that there exists a membership coefficient which corresponds to the degree to which an element belongs to a specific cluster. That is the point closer to the centre of a cluster has a high degree of membership than the points in the boundaries of a cluster. Consider an example

For example, a leaf can be green or yellow (hard clustering), but a leaf can also be green AND yellow (fuzzy clustering). Here, the leaf can be green to a certain degree as well as yellow to a certain degree. Instead of the leaf belonging to green [green = 1] and not yellow [yellow = 0], the leaf can belong to green [green = 0.5] and yellow [yellow = 0.5]. Although these values are 0 and 1 they do not represent probabilities and the sum of these two values won't necessarily add to 1.

The fuzziness of a clustering can be explained by the Dunn's partition coefficient  $F(k)$  of the clustering where  $k$  represents the total number of clusters. Dunn's partition coefficient is defined as the sum of squares of all membership coefficients, divided by the total number of data points. The value of  $F(k)$  ranges between  $k$  and  $\frac{1}{k}$ . A low value for the Dunn's coefficient indicates a very fuzzy clustering, whereas a value close to 1 indicates a near-crisp clustering.

The key difference between other form of clustering we have seen earlier and fuzzy clustering is that in fuzzy clustering a data point can exist in multiple clusters, whereas in  $k$  means or  $k$  medoid clustering, a data point can be associated to only one cluster.

One of the most popular fuzzy clustering techniques is fuzzy c-means (FCM) technique. In FCM the cluster centroid is calculated as the mean of all data points, weighted by their degree of belonging to the cluster

As we discussed earlier the aim of fuzzy c-means clustering algorithm is to minimize the cost function given below

$$\sum_{j=1}^k \sum_{x_i \in C_j} u_{ij}^m (x_i - \mu_j)^2$$

Where  $u_{ij}$  denotes the degree to which a data point  $x_i$  belongs to a cluster  $c_j$  and  $\mu_j$  represents the centre of cluster  $j$  and  $m$  is called the fuzzifier. It is observable that fuzzy c means algorithm is different from  $k$  means algorithm by the introduction of membership values  $u_{ij}$  and fuzzifier  $m$

We can define  $u_{ij}^m$  as below

$$u_{ij}^m = \frac{1}{\sum_{i=1}^k \left( \frac{|x_i - c_j|}{|x_i - c_k|} \right)^{\frac{2}{m-1}}}$$

From the above equation it is clear that the degree of belonging  $u_{ij}$  has an inverse link to the distance from data point  $x$  to cluster centre  $c$ . The parameter  $m$  represents the fuzziness of the clustering,  $m$  can have values between 1 and  $\infty$ . If the value of  $m$  is closer to 1, that clustering is very similar to  $k$ -means clustering, whereas the value of  $m$  close to  $\infty$  leads to complete fuzziness in the clustering.

The centroid of a cluster in fuzzy c means is given by the ratio of mean of all points to the degree of belongings to the cluster. The centroid  $C_j$  is given by the equation

$$C_j = \frac{\sum_{x \in c_j} u_{ij}^m x}{\sum_{x \in c_j} u_{ij}^m}$$

Here  $C_j$  is the centroid of the cluster and  $u_{ij}$  denotes the degree to which a data point  $x_i$  belongs to  $C_j$ . Now let us see the algorithm for fuzzy c means clustering.

### Algorithm

Step 1: Specify the number of clusters required( $k$ )

Step 2: Assign coefficient values randomly to the data points for being in the clusters.

Step 3: Compute the centroid for each cluster using the formula for  $C_j$

Step 4: Compute the membership coefficient for each point using formula for  $u_{ij}^m$

Step 5: Repeat step 3 and step 4 until the value for the coefficients converge

Step 6: Stop

## Python Program

```
# import required libraries numpy and FCM

from fcmeans import FCM
import numpy as np

# load the data points to X

X = np.array([[1, 1], [2, 1], [4, 3],[5, 4]])

# perform fcm clustering for n=2

fcm= FCM(n_clusters=2, random_state=0).fit(X)

# printing cluster centres

fcm_centers = fcm.centers

# finding label value having maximum membership value u

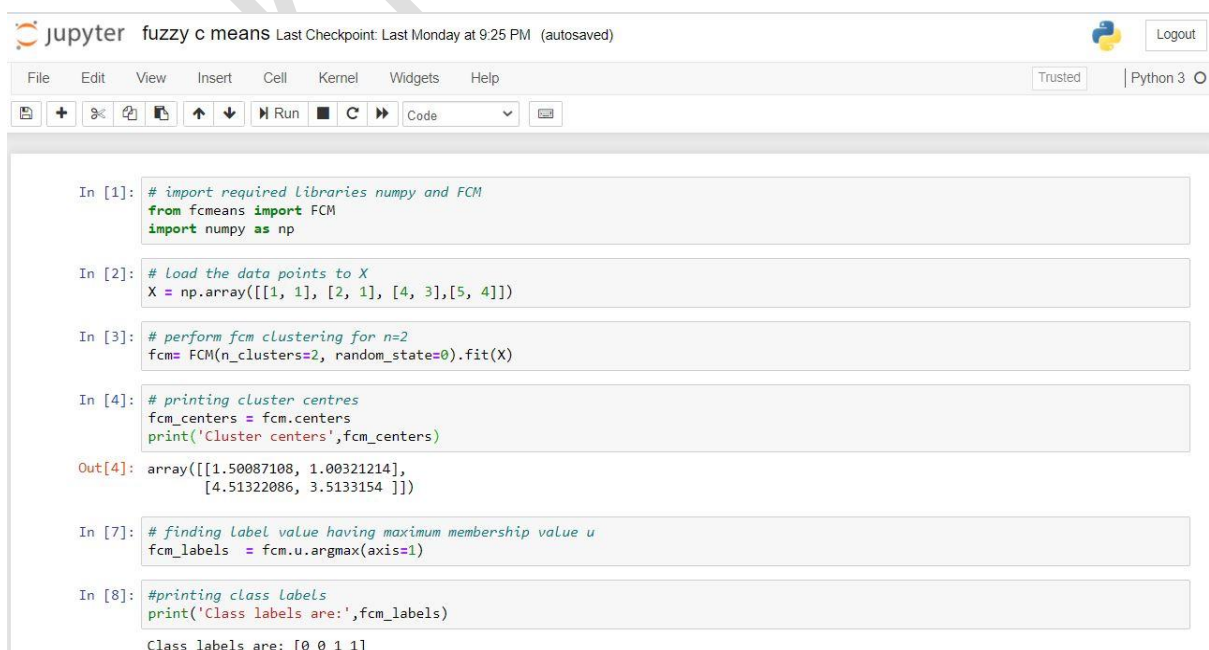
fcm_labels = fcm.u.argmax(axis=1)

fcm_labels #printing class labels
```

### Output:

```
array([0, 0, 1, 1], dtype=int64)
```

## Jupyter Notebook Screenshot



The screenshot displays a Jupyter Notebook window titled "fuzzy c means". The interface includes a top bar with the Jupyter logo, the title, and a "Last Checkpoint" timestamp. Below this is a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running cells, and other functions. The notebook content consists of several input cells (In [1] to In [8]) and one output cell (Out [4]).

```
In [1]: # import required libraries numpy and FCM
from fcmeans import FCM
import numpy as np

In [2]: # Load the data points to X
X = np.array([[1, 1], [2, 1], [4, 3],[5, 4]])

In [3]: # perform fcm clustering for n=2
fcm= FCM(n_clusters=2, random_state=0).fit(X)

In [4]: # printing cluster centres
fcm_centers = fcm.centers
print('Cluster centres',fcm_centers)

Out[4]: array([[1.50087108, 1.00321214],
               [4.51322086, 3.5133154 ]])

In [7]: # finding Label value having maximum membership value u
fcm_labels = fcm.u.argmax(axis=1)

In [8]: #printing class labels
print('Class labels are:',fcm_labels)

Class labels are: [0 0 1 1]
```

### 3.5. DBSCAN

DBSCAN algorithms (density-based spatial clustering of applications with noise) are especially used for clustering large datasets. The idea behind DBSCAN is to create clusters having a minimum number of points and density. Density is a measure of minimum number of data points within a certain distance of each other points. This idea helps to exclude the cluster formation by an outlier. The minimum number of points in any cluster need to be defined initially, let us denote that parameter with *MinPts*. *Eps* is another parameter which defines the threshold distance to maintain between data points with in a cluster. The *Eps*-neighborhood or neighborhood of a data point is defined as the set of points which falls within a distance of *Eps*. In DBSCAN algorithm, the number of clusters in the output is not pre-defined instead the algorithm determine the number of clusters ( $k$ )

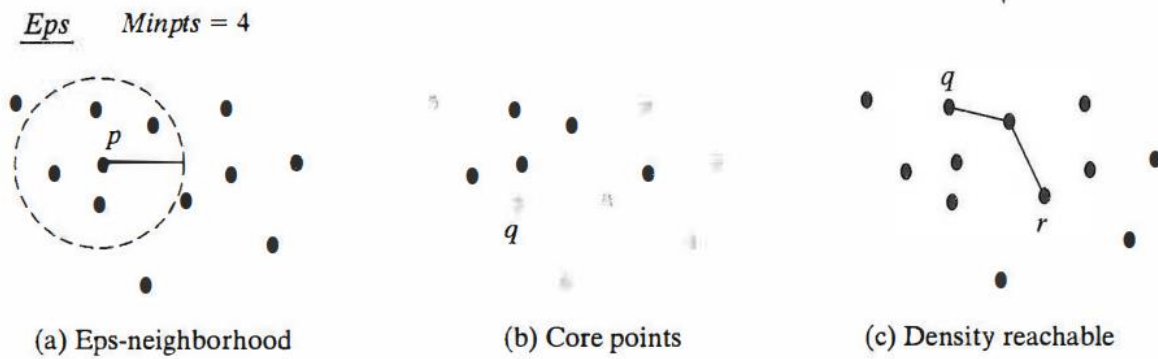
In order to understand DBSCAN and its operations, one must understand the concept of density. A data point  $p$  is said to be directly density reachable from data point  $q$ , if

$$\begin{aligned} &dis(p, q) \leq Eps \text{ and} \\ &|\{r | dist(r, q) \leq Eps\}| \geq MinPts \end{aligned}$$

The first part ensure that the data points  $p$  and  $q$  lies within the acceptable range *Eps*. The second part of the definition is to ensures that there are a minimum number of core points close to each other *MinPts*. These core points contribute to the main portion of the cluster. A directly density-reachable point must be dose to one of these core points, but it need not be a core point itself. In that case, it is called a border point. Two points are said to be density-reachable only if there exist and chain of directly density-reachable points from one to another. This property ensure that any cluster will have a core data points very close to a larger number of other data point (core points) and then some border points that are close to at least one core point. Let us try to explain the concept of *Eps* Neighbourhood, Core points and Density reachable with an example figure.

In the first part of the figure (*part a*), the point  $p$  has 4 data points with in the neighbourhood, since the value of *MinPts* is 4 the data point  $p$  is considered as a core

point. In the second part of the figure (*part b*), there are five core points and note that out of the 4 points that are in the neighbourhood of  $p$ , only 3 are themselves core points. These 4 points are said to be directly density-reachable from  $p$ . Point  $q$  is not a core point and is thus called a border point. We have partitioned the points into a core set of points that are all close to each other; then border points, which are close to at least one of the core points; and finally the remaining points, which are not close to any core point. *Part(c)* shows that even though point  $r$  is not a core point, it is density-reachable from  $q$ .



### Algorithm

Step 1: Consider  $X = (t_1, t_2, \dots, t_n)$  be the data points to be clustered.

Step 2: Initialize  $Eps$  and  $MinPts$  where  $Eps$  and  $MinPts$  are number of points in cluster and maximum distance for density measure respectively

Step 3: Consider the output  $K$  as the set of clusters  $K = (K_1, K_2, \dots, K_k)$

Step 4: Initialize  $k = 0$

Step 5: For each  $i = 1, 2, 3, \dots, n$ , if  $t_i$  is not in a cluster then  $X = \{t_j\}$  such that  $t_j$  is density reachable from  $t_i$

Step 6: If the generate  $X$  is a valid cluster, then increase the value of  $k$  by one and include  $X$  in the cluster  $K_k$

## Python Program

```
# import required libraries numpy and DBSCAN

from sklearn.cluster import DBSCAN

import numpy as np

# load the data points to X

X = np.array([[1, 1], [2, 1], [4, 3],[5, 4]])
# perform DBSCAN clustering for eps=2 and n=2

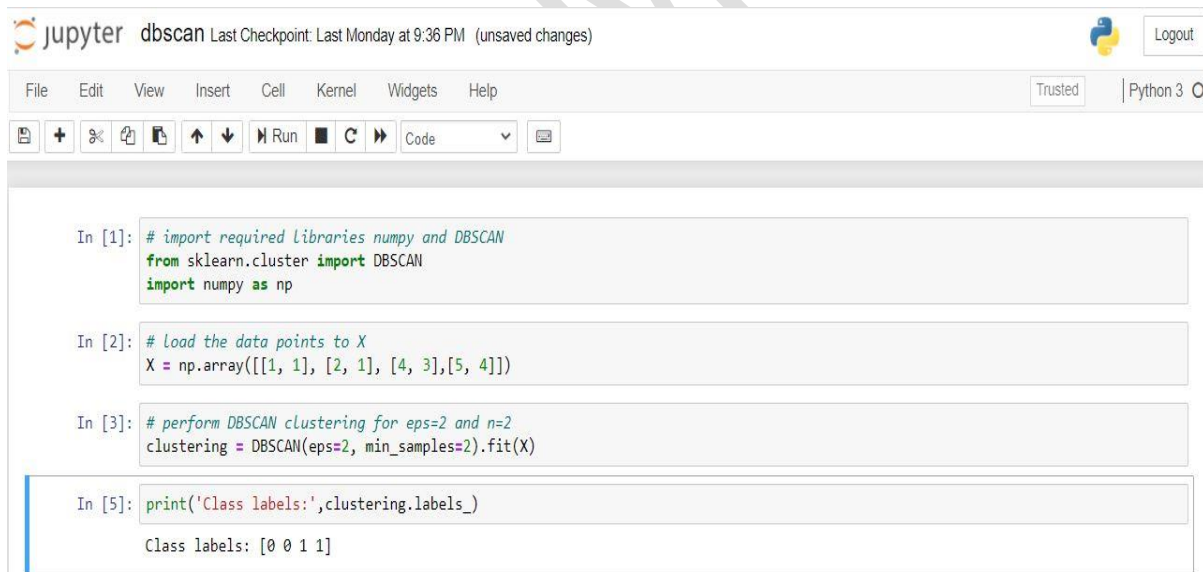
clustering = DBSCAN(eps=2, min_samples=2).fit(X)

clustering.labels_#printing class labels
```

### **Output:**

```
array([0, 0, 1, 1], dtype=int64)
```

## Jupyter Notebook Screenshot





## CHAPTER 4

### NATURAL LANGUAGE PROCESSING

#### 4.1. NATURAL LANGUAGE PROCESSING

The way humans express information is completely unstructured, Human beings generally communicate in words and sentences rather than tables, lists or excel sheets. Since humans writing and speaking unstructured the, complexity to understand it by the machines and computers is very high. Natural Language Processing (NLP) deals with this problem, it helps computers and machines to understand the unstructured text and vocals to retrieve meaningful pieces of information from it. You can consider Natural Language Processing (NLP) as a sub field of Artificial Intelligence (AI) which helps computers interact with humans.

##### Applications of NLP

- Speech Recognition
- Language Translation
- Automated Question Answering
- Automated Text Summary Generation
- Chatbot
- Spelling and Grammar check
- Autosuggestions and Autocomplete
- Email Spam detector

Although humans are very intelligent, we often misunderstand the context while interacting to one another, and we often interpret similar sentence or words completely differently. We can examine this scenario using a very simple example sentence “Call me a taxi, please” On can interpret the sentences in many ways and the sentence is ambiguous. The meaning can be understood in following ways.

- Call the person as a taxi
- Book a taxi to the person

Another example of ambiguous sentence is “I saw a man on a hill with a telescope.”

This sentence can be interpreted in many ways, such as

- There is a man standing on the hill and I saw him with a telescope.
- There is a man standing on the hill who has a telescope.
- I was standing on a hill and saw a man using my telescope.
- I was standing on a hill and saw a man having a telescope.
- There is a man on a hill, and I saw him something with my telescope.

These examples illustrate that language processing is not “deterministic”, that means the same language or sentence can have multiple interpretations and something convenient for one person can be inconvenient or not suitable for another person. Interpretations and suggestion can be varied from entity to entity. Since there is such ambiguity, we can consider Natural Language Processing (NLP) as a non-deterministic approach. Since Natural Language Processing (NLP) is non-deterministic, it can be used to create a highly intelligent machine, which is capable to understand how humans think and interpret in various contexts.

#### **4.1.1. Types of Natural Language Processing**

There are two different classification in Natural Language Processing, they are Rule-based Natural Language Processing and Statistical Natural Language Processing

##### Rule-based Natural Language Processing

Rule-based approaches are the first developed techniques to process Natural Language. Since these techniques are here for so long, they are proven to work very well. Some of the examples for Rule based NLP are Context Free Grammar (CFG) and Regular expressions. This technique uses common sense reasoning to operate or you can consider as “fill in the blanks” method to solve problems. The problems with Rule based NLP are the performance degradation occurring in generalized scenarios, and the approach takes much time and require manual efforts.

## Statistical Natural Language Processing

Statistical Natural Language Processing operates on large amounts of data and tries to find solutions from it. In Statistical NLP, Machine learning algorithms are used to train Natural Language Processing models. After the NLP model is trained successfully on large instance of data, the trained model can be used to solve NLP problems. Following table shows the key features and limitations of both methods

<b>Rule Based Natural Language Processing</b>	<b>Statistical Natural Language Processing</b>
Flexible	Easy to Scale
Easy to Operate	Self-Learning
No need of much training	Fast development
High precision	Can be used for Generalised problems
Needs highly skilled developers	Fast operation but much time for training
Slow operation	Require Large amount of data for training
Cannot applied on Generalised problems	Difficult to code and debug

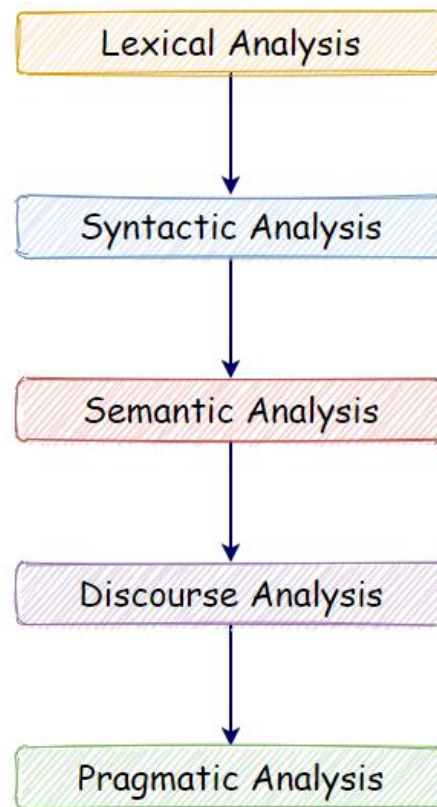
### **4.1.2. Components of Natural Language Processing (NLP)**

There are different stages in natural language processing as shown in below figure, they are Lexical Analysis, Syntactic Analysis, Semantic Analysis, Discourse Analysis and Pragmatic Analysis. We will discuss each of them one by one.

#### **a. Lexical Analysis**

Lexical analysis otherwise called as structure analysis here we analyse the structure of the sentence. Lexicon of a language means the collection of words and phrases in a language. The lexical analysis phase divides the whole input text to paragraphs,

sentences, and words. It involves identifying and analysing words' structure, and hence known as structure analysis.



b. Syntactic Analysis

Syntactic analysis is otherwise called as parsing, in this stage the words (output of lexical analysis phone) are analysed for grammar and are arranged in a fashion that shows the relationship among the words. Consider an example sentence, “The shop goes to the garden” does not get accepted by English syntactic analyser, since it does not show any relationship between words.

c. Semantic Analysis

Semantic analysis tries to find the accurate meaning for the words, and it analyses the text meaningfulness. This stage make sure that sentences such as “hot ice-cream” do not go through the Semantic checking phase.

d. Disclosure Integration

Disclosure integration takes into account the context of the text. It considers the meaning of the sentence before it ends. Consider an example “He came to office” In this case, “he” must be referenced in preceding sentence.

e. Pragmatic Analysis

Pragmatic analysis is the last stage of Natural Language Processing. The overall communication and interpretation of language is carried out in Pragmatic analysis stage. It aims to develop meaningful use of language in various situations.

### **4.1.3. Stemming**

Stemming is a procedure in Natural Language Processing used to normalize words. In languages, a single word can have multiple forms depending upon the context used. Let us consider an example word “carry”, The word “study” is a verb, but it can take many other forms such as “carries,” “carried,” “carrying” and so on, related to its context. During the lexical analysis phase, the input text or speech is tokenized and the interpreter considers these input words as different words even though they convey same meaning. Since the goal of Natural Language Processing is to understand the meaning of content for different context, we use stemming to resolve this problem.

The stemming procedure normalize the word by truncating the word to its stem word. Let us see one example, the words “carries,” “carried,” “carrying” will be reduced to “carri” and all these words are referred by the stem word “carri” only. That is the token used for all the word forms is “carri”. The stemmed output “carri” does not carry any meaning or included in dictionary, Stemming may not give us a dictionary, grammatical word for a particular set of words. There are different types of Stemming such as Porter’s Stemmer, Snowball Stemmer, Lovin’s Stemmer and many more. One advantage of Snowball Stemmer is that it supports a wide variety of languages.

#### 4.1.4. Lemmatization

Lemmatization operates similarly to stemming, and tries to find a similar base “stem” for a given word. But the Lemmatization differ from stemming, Lemmatization finds a dictionary word as a base “stem” instead of truncating the original word. Lemmatization is slower compared to stemming, because stemming does not consider the context or meaning of the stemmed word. Lemmatization is slower but more accurate compared to stemming.

The choice of stemming or lemmatization depends on the requirement and project goal. If a project needs faster responses, then stemming may be used, but If higher accuracy is crucial, then the best option is Lemmatization.

Lemmatization takes into account Part of Speech (POS) values, The process may produce different outputs for different values of POS. We generally have four options for POS, such as Verb(v), Noun(n), Adjective(a) and Adverb()

#### *Python Program- Showing difference between Stemming and Lemmatization*

```
# import required libraries nltk, PorterStemmer, WordNetLemmatizer

import nltk
nltk.download('wordnet')
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer

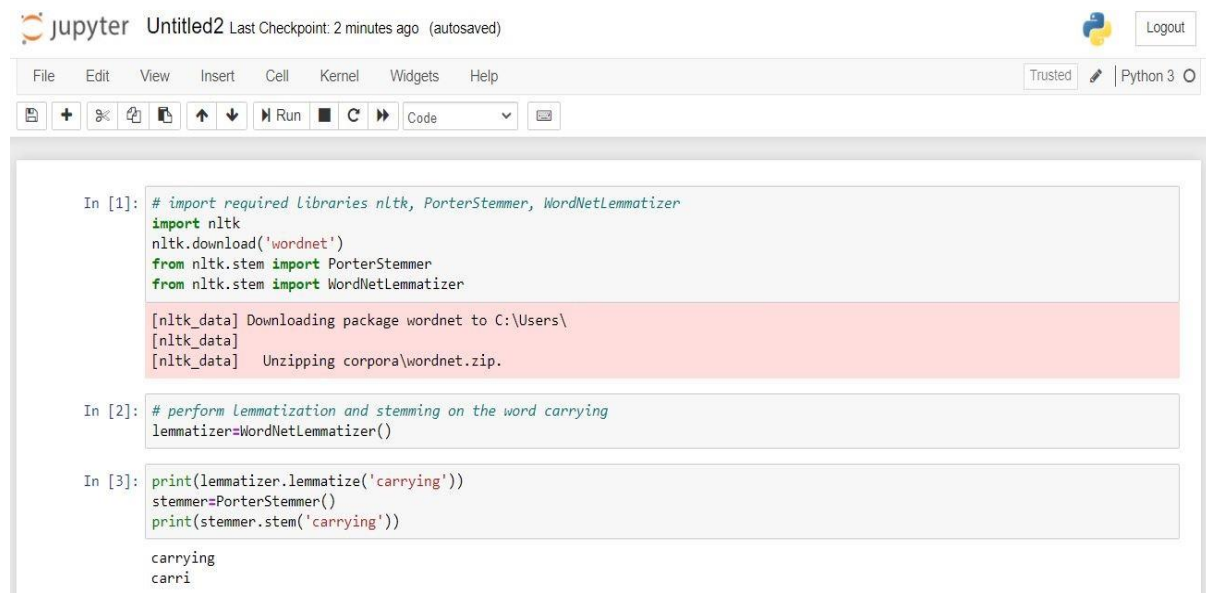
# perform lemmatization and stemming on the word carrying

lemmatizer=WordNetLemmatizer()
print(lemmatizer.lemmatize('carrying'))
stemmer=PorterStemmer()
print(stemmer.stem('carrying'))
```

#### **Output:**

```
Carrying
Carri
```

## Jupyter Notebook Screenshot



```
In [1]: # import required libraries nltk, PorterStemmer, WordNetLemmatizer
import nltk
nltk.download('wordnet')
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer

[nltk_data] Downloading package wordnet to C:\Users\
[nltk_data]
[nltk_data] Unzipping corpora\wordnet.zip.

In [2]: # perform lemmatization and stemming on the word carrying
lemmatizer=WordNetLemmatizer()

In [3]: print(lemmatizer.lemmatize('carrying'))
stemmer=PorterStemmer()
print(stemmer.stem('carrying'))

carrying
carri
```

### 4.1.5. Part of Speech Tagging (PoS tagging)

Let us discuss the we need Part of Speech (PoS). Consider the simple question “can you help me with the can?” The word “can” can have several meanings depending on the context and PoS. The first occurred “can” is used as a question word for question formation. The second “can” at the end of the sentence is intended to denote a container or bucket. If you look at the Pat of Speech (PoS), the first “can” is a verb, and the second “can” is a noun. Giving the word a specific meaning allows the program to handle it correctly in both semantic and syntactic analysis and hence PoS tagging is important for syntactic and semantic analysis to solve sentences which has several semantic meanings.

#### Python Program

```
# import required libraries nltk to perform pos tagging

import nltk
nltk.download('averaged_perceptron_tagger')

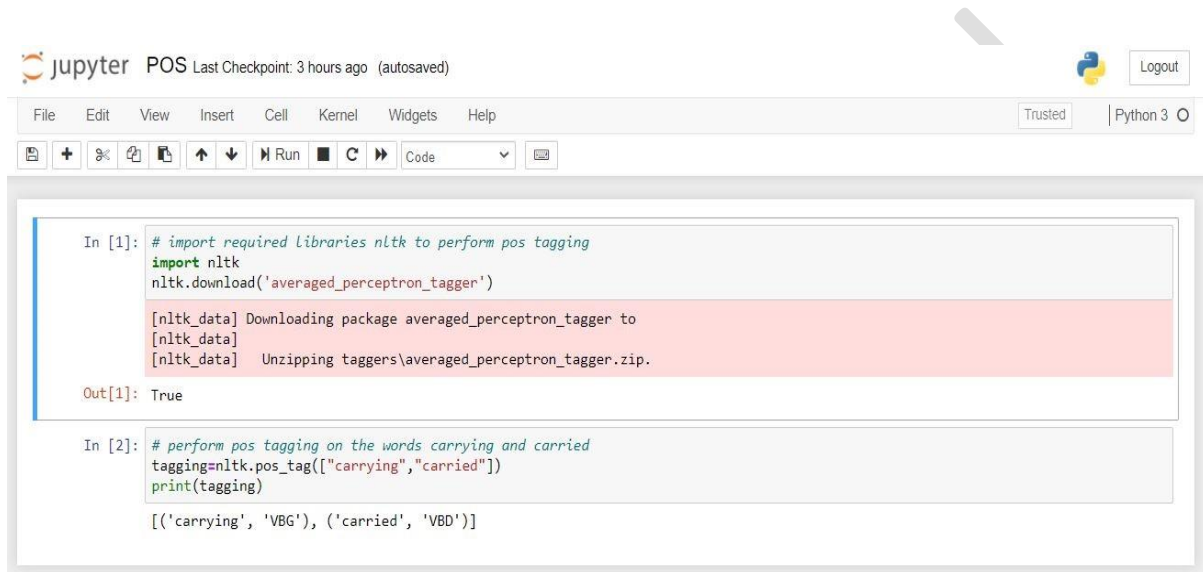
# perform pos tagging on the words carrying and carried
```

```
tagging=nlk.pos_tag(["carrying","carried"])
print(tagging)
```

### **Output:**

```
[('carrying', 'VBG'), ('carried', 'VBD')]
```

### **Jupyter Notebook Screenshot**



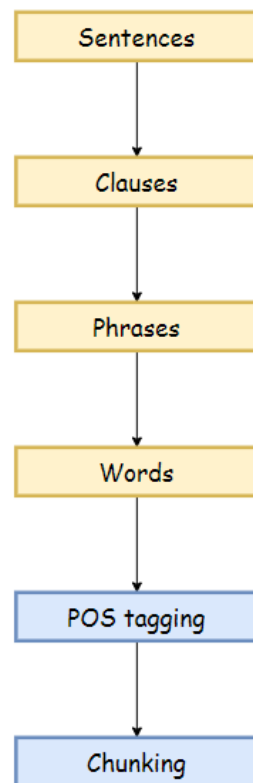
### **4.1.6. Chunking**

Chunking is a procedure applied to extract meaningful phrases or group of words from unstructured text. Chunking works on top of PoS tagging. We cannot find meaning full information from tokens only. Tokens are passed through PoS tagging stage and then fed to Chunking phase as shown in above figure. Chunking means a group of words, which breaks simple text into phrases that are more meaningful than individual words.

Now let us understand, what are the output of chunking phase, we have mentioned that chunking produces meaningful groups of words are called phrases. What are these words, let us see. There are five categories of phrases such as Noun Phrases (NP), Verb Phrases (VP), Adjective Phrases (ADJP), Adverb Phrases (ADVP), Prepositional Phrases (PP).



There are different rules to construct these phrases, they are given below in the form of a grammar notation.



### Phrase structure rules

- $S(\text{Sentence}) \rightarrow NP VP$
- $NP \rightarrow \{\text{Determiner, Noun, Pronoun, Proper name}\}$
- $VP \rightarrow V (NP)(PP)(\text{Adverb})$
- $PP \rightarrow \text{Pronoun } (NP)$
- $AP \rightarrow \text{Adjective } (PP)$

Consider an example sentence “We carried a white rabbit”, where “we” is noun, carried is verb, a is article, white is adjective and rabbit is pronoun.

#### 4.1.7. Named Entity Recognition (NER)

Named entity recognition (NER) is used to automatically scan entire text and pull out some fundamental information's such as date, time, people, organizations, places, money etc. Applications of NER include summarization of resume, Automatic plot creation for movies, Content extraction for news agencies, Search engine optimization etc...

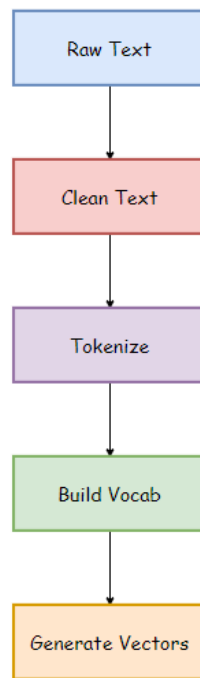
Let us see some commonly used entities in the below table

Named Entity	Example
Date	25-10-2020
Time	10.12 pm
People	Robert De Niro
Organization	NASA
Place	Washington
Money	1000 USD
Percent	30%

#### 4.1.8. Bag-of-Words method

In this method, raw text is analysed to extract meaningful features, it is said to be bag of words since the words are mixed up that is there is no order in occurrence of these words. This model transforms the given text into words, and it also record the frequency of the words in the text.

In short, you can consider bag words as a representation of a sentence as a collection of words with frequency of occurrence with no given importance to the order.



- a. Raw Text: This is the input text on which we operate
- b. Clean Text: Unnecessary characters such as punctuation marks and stop words are removed to clean up the text.
- c. Tokenize: The sentence is represented as a group of tokens or words.
- d. Building Vocab: This contains total words used in the text after removing unnecessary data.
- e. Generate Vocab: It contains the words along with their frequency of occurrence in the sentences.

Let us create Bag of words for the below given sentences.

- a. Arun and athul travelled by the flight.
- b. The train was delayed.

Here the first step is to create a basic structure by removing punctuations, full stop and other unnecessary characters.

Sentence 1	Sentence 2
arun	the
and	train
athul	was
travelled	delayed
by	
the	
flight	

Now words frequencies are recorded

Sentence 1	Frequency	Sentence 2	Frequency
arun	1	the	1
and	1	train	1
athul	1	was	1
travelled	1	delayed	1
by	1		
the	1		
flight	1		

Now sort words in alphabetic and combine sentence 1 and sentence 2

Sentence	Frequency
And	1
Arun	1
Athul	1
By	1
Delayed	1
Flight	1
The	2
Train	1
Travelled	1
was	1

Now the final bad of words would look line this

Words	Sentence-1	Sentence-2
And	1	0
Arun	1	0
Athul	1	0
By	1	0

Delayed	0	1
Flight	1	0
The	1	1
Train	0	1
Travelled	1	0
was	0	1

#### 4.1.9. Term Frequency — Inverse Document Frequency (TF-IDF)

Term Frequency — Inverse Document Frequency is word scoring measure used for information retrieval and summarization. The term is used to express the importance of a word in a given text. The word which appears multiple times in a document can be considered as a highly important word compared to other words that appear fewer times. If a word appears many times in a document and the same word appear frequently in some other documents also, then the word can be considered as a general occurring word and we cannot assign much importance to it.

Suppose we want to search for “quality mobile” and consider the search is done on a large database having mobile descriptions. The search program is designed to display the closest response to the user query. The search engine will compute the TF-IDF to calculate the score for all of our available descriptions in the database, and the result with the higher score will be retrieved as search result. If there is an exact match for the user query, there is no need for TF-IDF calculation, and the exactly matched description is shown as result.

Suppose we have following description in our database

- a. The good mobile
- b. a quality cell
- c. a fast mobile
- d. the cheap mobile cell phone

The description a has one word out for three words in user query, description c has one-word matching user query also description d has one-word matching user query. But description b has one word “quality” which is an essential word. So the search program will display description b as the search result.

Now if you look at the descriptions, you can see that the word “mobile” is common in the descriptions, and therefore it will have a very lesser IDF value and thereby a lower TF-IDF value. If you consider the word “quality” it appears rarely in the other descriptions and have a high discriminating power compared to the word “mobile”, hence the value of TF-IDF will be higher, and the search program can conclude the result as description b. TF\*IDF score is directly proportional to the importance of the term, that is higher the TF\*IDF score, valuable the term and vice versa

#### Equations for TF, TF-IDF, IDF

- a. TF(Term Frequency)

$$TF = \frac{\text{Frequency of the term in the sentence}}{\text{Total number of terms in the sentence}}$$

- b. IDF(Inverse Document Frequency)

$$IDF = \frac{\text{Total number of sentences}}{\text{Number of sentences containing that term}}$$

- c. TF-IDF(Term Frequency — Inverse Document Frequency)

$$TF - IDF = TF * IDF$$

## Python Program-TF-IDF

```
# import required libraries NLTK and modules sent_tokenize, word_tokenize and
FreqDist and you may require to use- nltk.download('punkt')
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
# input the sentences for which TFIDF need to be found
```

```
sent=['to do is to be to be is to do','to be or not to be i am what i am','i think
therefore i am do be do be do','do do do da da da let it be let it be']
```

```
# generating TFIDF to Output
```

```
vector=TfidfVectorizer(norm= None)
Output=vector.fit_transform(sent).toarray()
Print (vector.get_feature_names()) #printing vocabulary
print(Output) #printing TFIDF
```

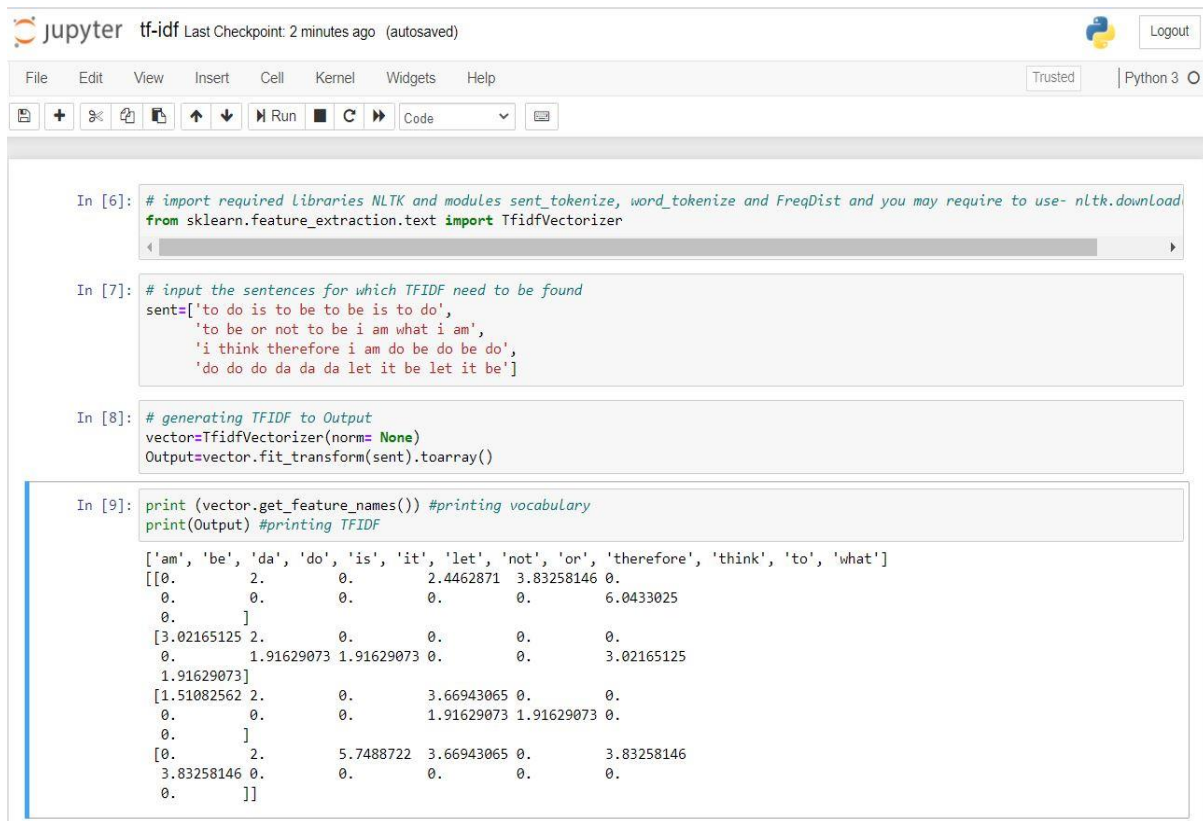
### Output:

```
['am', 'be', 'da', 'do', 'is', 'it', 'let', 'not', 'or', 'therefore',
'think', 'to', 'what']
```

```
[[0.         2.         0.         2.4462871  3.83258146 0.
  0.         0.         0.         0.         0.         6.0433025
  0.         ]
 [3.02165125 2.         0.         0.         0.         0.
  0.         1.91629073 1.91629073 0.         0.         3.02165125
  1.91629073]
 [1.51082562 2.         0.         3.66943065 0.         0.
  0.         0.         0.         1.91629073 1.91629073 0.
  0.         ]
 [0.         2.         5.7488722  3.66943065 0.         3.83258146
  3.83258146 0.         0.         0.         0.         0.

  0.         ]]
```

## Jupyter Notebook Screenshot



```
In [6]: # import required libraries NLTK and modules sent_tokenize, word_tokenize and FreqDist and you may require to use- nltk.download
from sklearn.feature_extraction.text import TfidfVectorizer

In [7]: # input the sentences for which TFIDF need to be found
sent=['to do is to be to be is to do',
      'to be or not to be i am what i am',
      'i think therefore i am do be do be do',
      'do do da da da let it be let it be']

In [8]: # generating TFIDF to Output
vector=TfidfVectorizer(norm=None)
Output=vector.fit_transform(sent).toarray()

In [9]: print(vector.get_feature_names()) #printing vocabulary
print(Output) #printing TFIDF

['am', 'be', 'da', 'do', 'is', 'it', 'let', 'not', 'on', 'therefore', 'think', 'to', 'what']
[[0. 2. 0. 2.4462871 3.83258146 0.
 0. 0. 0. 0. 0. 6.0433025
 0.
 ]
 [3.02165125 2. 0. 0. 0. 0.
 0. 1.91629073 1.91629073 0. 0. 3.02165125
 1.91629073]
 [1.51082562 2. 0. 3.66943065 0. 0.
 0. 0. 0. 1.91629073 1.91629073 0.
 0.
 ]
 [0. 2. 5.7488722 3.66943065 0. 3.83258146
 3.83258146 0. 0. 0. 0. 0.
 0.
 ]]
```

## 4.2. LIBRARIES FOR NATURAL LANGUAGE PROCESSING

Here we will see a number of python libraries for NLP

### a. Natural Language Toolkit(NLTK)

NLTK Python framework is a very famous NLP tool. It is not only used for education and research but also used in production environments. It is very easy and free to use tool .

Features: Tokenization, Part Of Speech tagging, Named Entity Recognition, Sentiment analysis



b. spaCy

spaCy is another widely used open-source natural language processing library available for python, It is very fast and optimal and used mainly for production environments.

Features: Tokenization, Part Of Speech tagging, Named Entity Recognition, Sentiment analysis, Dependency parsing

c. TextBlob

TextBlob is a python library used for processing large textual data.

Features: Part-of-Speech tagging, Noun phrase extraction, Sentiment analysis, Classification, Language translation, Wordnet integration.

d. Gensim

Gensim is another NLP python framework used for similarity detection. Although it is not a general-purpose NLP library, it handles the assigned tasks very well.

Features: Latent semantic analysis, Non-negative matrix factorization, TF-IDF

e. Pattern

It is an NLP python framework with straightforward syntax. It's is used for scientific research purpose and production. It is a very powerful and valuable tool for NLP researchers and students.

Features: Tokenization, Part of Speech tagging, Named entity recognition, Parsing, Sentiment analysis

Example python program to explore features of NLTK python library

```
# import required libraries NLTK and modules sent_tokenize, word_tokenize and  
FreqDist and you may require to use- nltk.download('punkt')
```

```
import nltk
```

```
nltk.download('punkt')
```

```
from nltk import sent_tokenize
```

```
from nltk import word_tokenize
```

```
from nltk.probability import FreqDist
```

```
# read the text file NLP.txt (you may choose any text file)
```

```
input_file=open("NLP.txt")
```

```
text=input_file.read()
```

```
# nltk used to perform sentence tokenization, word tokenization and last to find  
the frequency distribution of text and to print most frequent 10 words
```

```
sentence=sent_tokenize(text)
```

```
word=word_tokenize(text)
```

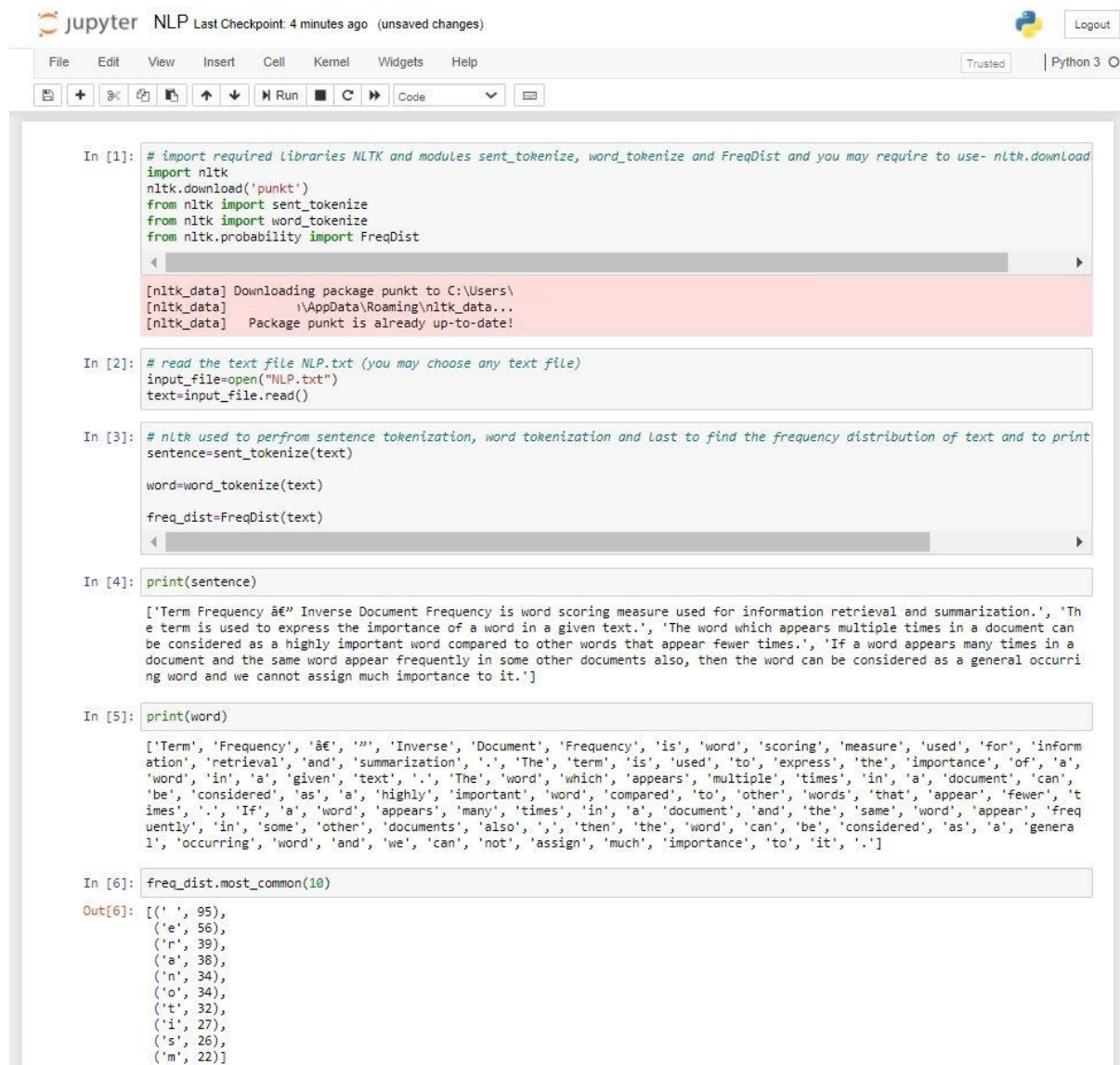
```
freq_dist=FreqDist(text)
```

```
print(sentence)
```

```
print(word)
```

```
freq_dist.most_common(10)
```

## Jupyter Notebook Screenshot



The screenshot shows a Jupyter Notebook interface with the title "NLP". The top bar indicates "Last Checkpoint: 4 minutes ago (unsaved changes)" and includes a "Logout" button. The interface has a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The notebook contains six input cells and one output cell.

```
In [1]: # import required Libraries NLTK and modules sent_tokenize, word_tokenize and FreqDist and you may require to use- nltk.download
import nltk
nltk.download('punkt')
from nltk import sent_tokenize
from nltk import word_tokenize
from nltk.probability import FreqDist

[nltk_data] Downloading package punkt to C:\Users\
[nltk_data]   \AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!

In [2]: # read the text file NLP.txt (you may choose any text file)
input_file=open("NLP.txt")
text=input_file.read()

In [3]: # nltk used to perform sentence tokenization, word tokenization and last to find the frequency distribution of text and to print
sentence=sent_tokenize(text)

word=word_tokenize(text)

freq_dist=FreqDist(text)

In [4]: print(sentence)

['Term Frequency &quot; Inverse Document Frequency is word scoring measure used for information retrieval and summarization.', 'The term is used to express the importance of a word in a given text.', 'The word which appears multiple times in a document can be considered as a highly important word compared to other words that appear fewer times.', 'If a word appears many times in a document and the same word appear frequently in some other documents also, then the word can be considered as a general occurring word and we cannot assign much importance to it.']

In [5]: print(word)

['Term', 'Frequency', '&quot;', '&quot;', 'Inverse', 'Document', 'Frequency', 'is', 'word', 'scoring', 'measure', 'used', 'for', 'inform', 'ation', 'retrieval', 'and', 'summarization', '.', 'The', 'term', 'is', 'used', 'to', 'express', 'the', 'importance', 'of', 'a', 'word', 'in', 'a', 'given', 'text', '.', 'The', 'word', 'which', 'appears', 'multiple', 'times', 'in', 'a', 'document', 'can', 'be', 'considered', 'as', 'a', 'highly', 'important', 'word', 'compared', 'to', 'other', 'words', 'that', 'appear', 'fewer', 'times', '.', 'If', 'a', 'word', 'appears', 'many', 'times', 'in', 'a', 'document', 'and', 'the', 'same', 'word', 'appear', 'frequently', 'in', 'some', 'other', 'documents', 'also', ',', 'then', 'the', 'word', 'can', 'be', 'considered', 'as', 'a', 'general', 'occurring', 'word', 'and', 'we', 'can', 'not', 'assign', 'much', 'importance', 'to', 'it', '.']

In [6]: freq_dist.most_common(10)

Out[6]: [(' ', 95),
 ('e', 56),
 ('r', 39),
 ('a', 38),
 ('n', 34),
 ('o', 34),
 ('t', 32),
 ('i', 27),
 ('s', 26),
 ('m', 22)]
```

## CHAPTER 5

### DEEP LEARNING

Artificial Neural Network is fairly new and efficient system whose key working idea is based on the biological neural networks. ANN also known as “artificial neural systems,” or “connectionist systems.” Artificial Neural Networks are composed of lots of small interconnected units called as nodes or neurons, and these units are capable to exchange information between them. These units can be considered as simple processors that has the ability to operate in parallel.

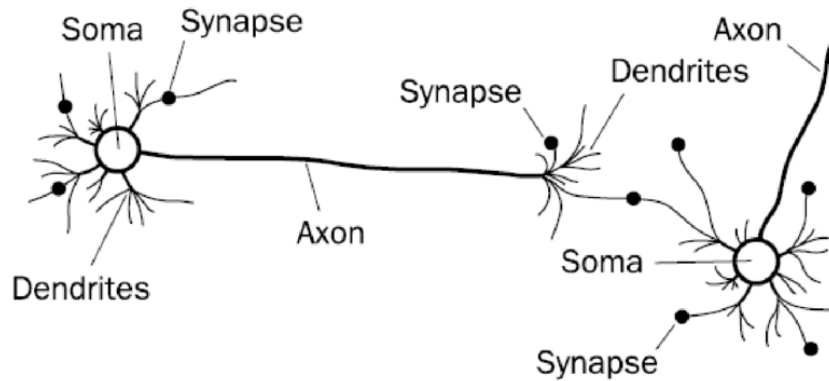
There exist a connection between each neurons, and the connections are associated with a weight. This weight bear the information about the input signal. The presence of weights are very important and a key factor for ANN, changes to these connection weights can excite or inhibit the input signal that is being communicated. Each neurons has an internal activation function, and only the input signals satisfying the activation function pass through the neuron. The output of such neurons are again fed as he input to the other linked neurons.

Now Let us consider the Biological Neuron for understanding the key principles.

#### 5.1. BIOLOGICAL NEURAL NETWORK

You can call a nerve cell as a neuron; these neurons are special type of cells seen in human body capable of processing and passing information to brain. Approximately there are around  $10^{15}$  neurons present in human body. The different parts of a biological neuron is shown in the below figure, now we can discuss their functionalities also.

1. Dendrites – These are branch like structure responsible for receiving information from nearby neurons.



2. Soma – This is the main cell body of the neuron which process information received from dendrites.
3. Axon – It is a connection link through which neurons send the information.
4. Synapses – It is the link between axon and dendrites of other nearby neurons

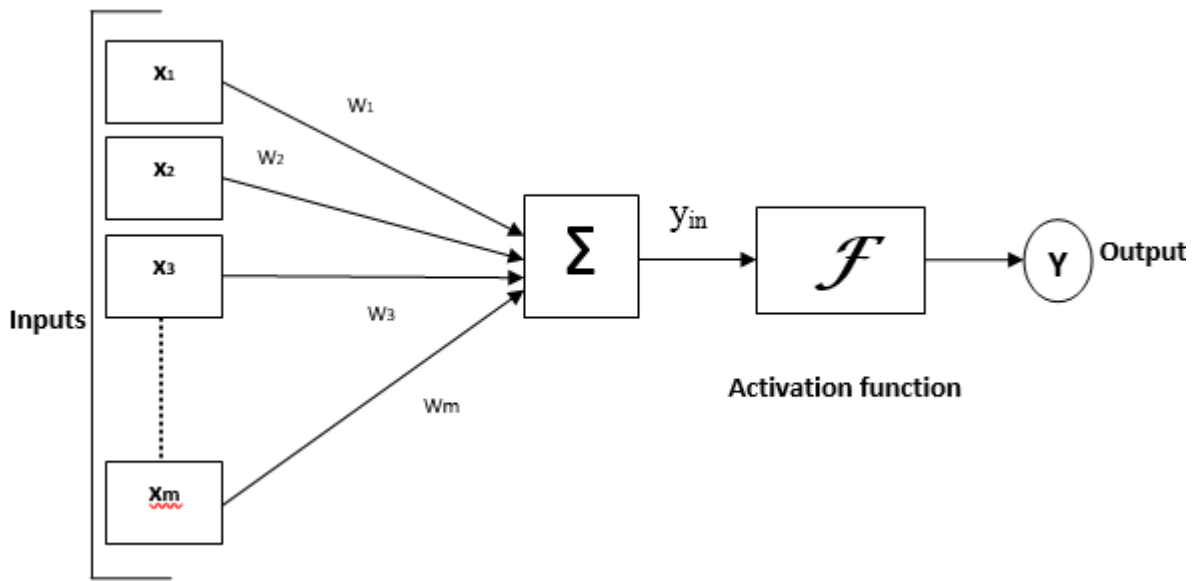
Now let us consider the difference between biological neural network and artificial neural network

Biological Neural Network	Artificial Neural Network
Processing is massively parallel and superior than ANN	Can be designed as massively parallel, but inferior than BNN
10 <sup>11</sup> neurons and 10 <sup>15</sup> interconnections	fewer neurons and interconnections
Degrade performance over time	Capable of fault tolerance
Stores information in synapse	The information is stored in assigned memory locations

Now let us see how an artificial neural network is designed by taking the idea of biological neural network. In the below section we will see a simple ANN model.

## 5.2. SIMPLE MODEL OF ARTIFICIAL NEURAL NETWORK

The below shown is a simple diagram of Artificial Neural Network.



Here the  $x_1, x_2, \dots, x_m$  represents the inputs given to a neuron, and the connection weights are given by  $w_1, w_2, \dots, w_m$ . Weighted summation of these input is computed and fed to the neuron as  $y_{in}$ . The activation function  $F$  will operate on this and produce and output  $Y$

That is

$$y_{in} = x_1 w_1 + x_2 w_2 + \dots + x_m w_m$$

$$y_{in} = \sum_{i=1}^m x_i w_i$$

The output  $Y$  can be calculated by applying the activation function over the net input  $y_{in}$  given as

$$Y = F(y_{in})$$

### 5.2.1. Types of Activation Functions

There are different type of activation function  $F$ , we will see these activation functions below.

#### a. Step function

This is one of the simplest form of activation function. In this type if the input is greater than a given threshold, then the neuron activates

$$f(x) = 1, \text{ if } x \geq t$$

$$f(x) = 0, \text{ if } x < t$$

#### b. Sigmoid function

Sigmoid function is another widely used activation function and given as the value of activation function ranges between 0 and 1. The activation function  $f(x)$  is given as

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

The advantage of sigmoid function is that it is continuously differentiable, and the function is non-linear.

#### c. Tanh function

The tanh function is another form of activation function and takes value between -1 and 1, the activation function  $f(x)$  is given as

$$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

#### d. ReLU

Rectified linear unit (ReLU) is the most widely used activation function and its value range between 0 and  $+\infty$ . ReLU activation function is given as

$$ReLU(x) = f(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

The advantage of ReLU function is that it will not activate all neurons at once, that means if the input to the neuron is negative, the output of ReLU function will be zero and the neuron won't get activated.

#### e. Leaky ReLU

Leaky ReLU (Linear Rectified Linear Unit) is a slight modification of ReLU, and thus form of activation function is defined as a linear multiple of input  $x$ . The value of Leaky ReLU ranges between  $-\infty$  and  $+\infty$ . Leaky ReLU is given as

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0.001x, & x < 0 \end{cases}$$

Artificial Neural Networks has wide applications in Classification, Clustering, Pattern matching, Control system, Optimization and Data mining.

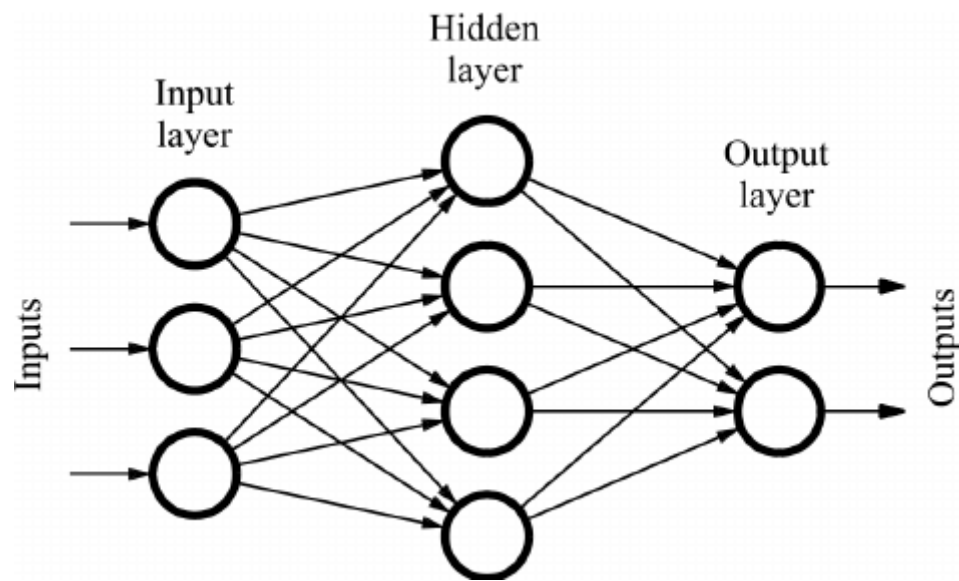
### 5.3. TYPES OF ANN

Now let us see different types of artificial neural networks, based on the architecture of neural network there are mainly 3 types. They are

- Feedforward Network
- Feedback Network
- Lateral Network



### 5.3.1. Feedforward Network

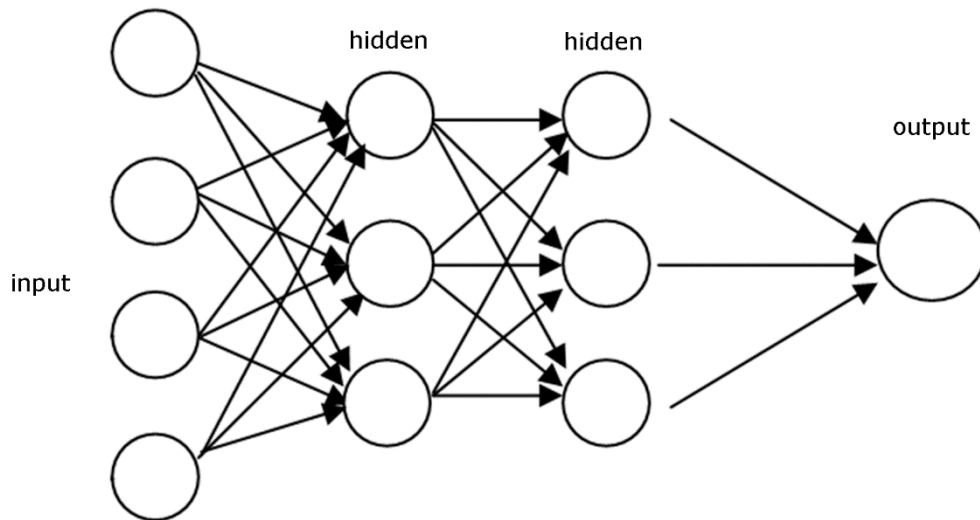


As shown in the above figure, there are three layers in this type of network, they are input layer, hidden/middle layer and output layer. The first layer is input layer and the number of neurons in this layer is proportional to the number of inputs. The input layer is said to be a passive layer since the neurons in this layer won't participate in processing the data, they just pass input to the next layers.

The second layer is hidden layer, and they receive inputs from the input layer. The number of neurons, and the layers in hidden layer is arbitrary and they vary according to the model design. Neurons in this layer are said to be active since they participate in data processing and signal modification.

The third layer of this architecture is output layer, they are also active layers. The number of neurons in this layer is proportional to the number of output values of the neural network.

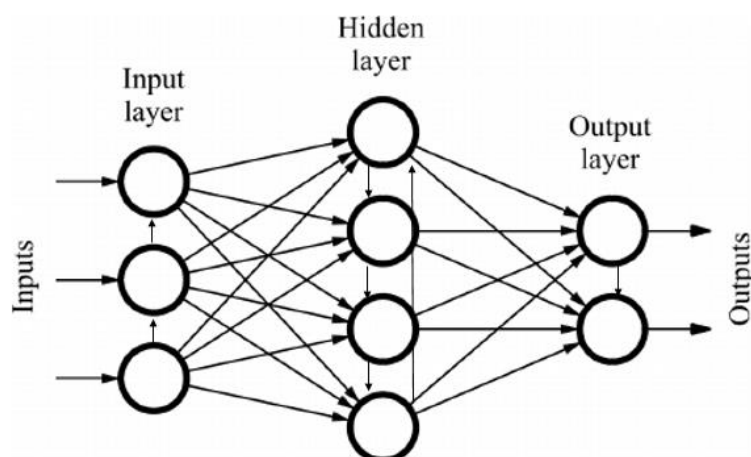
Feedforward networks can have multiple hidden layers depending on the model design, below is a figure of a feedforward network with two hidden layers.



### 5.3.2. Feedback Network

Feedback neural networks are the network in which feedbacks are present, that means the output of neuron in the networks is either directly or indirectly fed back to its input via other linked neurons. Feedback neural networks are otherwise called as Recurrent Neural Networks. LSTM (Long Short-Term Memory )is an example of RNN.

### 5.3.3. Lateral Network



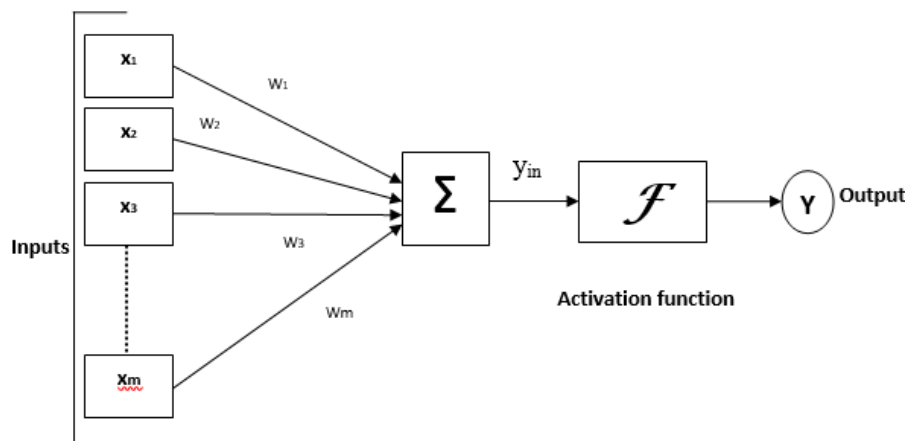
Lateral networks can be considered as a compromise between feed forward and feed back neural networks, There is no presence of designated feed back path in this type of networks, but there exist coupling of neurons with in a layer. A representation of lateral network is shown in above figure.

## 5.4. BACKPROPAGATION

Back-propagation is the key concept of training a neural network, the training of a neural network refers to the adjustment of weights of connection links in the network. A fine tuning of the weight leads to a better accurate model. Every iteration of backpropagation aims to achieve a better model than its predecessor. Back propagation is otherwise called as “backward propagation of errors”. The key idea of backpropagation is to find the gradient of a loss function with respects to all the weights in the network.

### Steps in Backpropagation

- Input  $X = \{x_1, x_2, \dots, x_m\}$  arrives
- Link weights are noted as  $\{w_1, w_2, \dots, w_m\}$ , these weights are initially random and our aim to adjust this weights for a better model
- The output  $Y$  is calculated
- Error is calculated as  $Errors = Desired\ output - Actual\ output\ (Y)$
- Go back to the hidden layers and adjust the weights so that the *Error* becomes minimum



$$Y = F(y_{in})$$

$$y_{in} = \sum_{i=1}^m x_i w_i$$

$$Y = F\left(\sum_{i=1}^m x_i w_i\right)$$

Error is given as

$$Error = \sum_{i=1}^c (t_c - F_c)^2$$

Where  $t_c$  and  $F_c$  represent the target output and actual obtained output, The equation for weight update is given below

$$w_{new} = w_{old} + \Delta w_{old}$$

$$\Delta w_{old} = \eta \left( -\frac{\partial E}{\partial w_{old}} \right)$$

Where  $\eta$  is the learning rate

$\frac{\partial E}{\partial w_{old}}$  is given as

$$\frac{\partial E}{\partial w_{old}} = -2(t_c - F_c) \frac{\partial F}{\partial y_c} x_{ic}$$

## 5.5. NEED FOR DEEP LEARNING

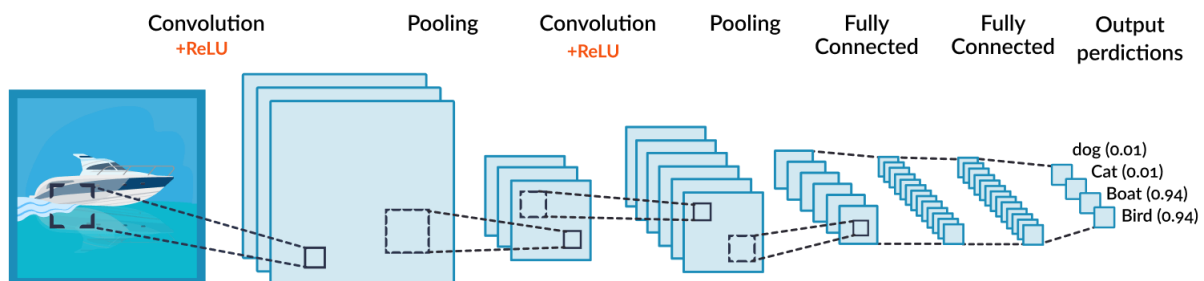
Deep learning models are often known as deep neural networks because most deep learning models are implemented using neural networks. The idea behind the term “deep”, is from the usage of more hidden layers in neural networks. A deep neural network may contain more than 100 hidden layers compared to two or three layers of normal neural networks.

A machine learning extract relevant features manually from data set and these features are used to create a model, but in deep learning relevant features are automatically extracted from dataset. Another important difference is that the deep learning is designed to use for large data sets and are usually scalable.

### 5.5.1. Convolutional Neural Networks

Convolutional neural networks (CNN or ConvNet) are one of the most popular deep neural network, the technique being used in many of the deep learning applications. CNN is suitable for 2-D data because, the operation in CNN is a 2-D convolution of extracted features with input data. CNN is widely being used for deep learning over image data.

The advantage of CNN is that, there is no need for manual feature extraction, the system is designed to extract feature directly from input data. You don't need to define and provide the relevant features. Since the feature extraction is automated CNN based deep learning models can be used for real time object detection and provide high accuracy for computer vision. Below given is a simple architecture of a convolutional neural network, there are a number of convolutional layers, optional pooling layers and fully connected layers. Let us see each layers in details below.



#### Convolution Layer

The fundamental of a CNN resides in the convolution operation. Hence they got the name convolutional neural networks. The first layer of a CNN is convolution, and this layer is being used to extract features from input data. Convolution stage convolve a feature map of an input hidden layer with a weight matrix and produce an output feature map. There are four dimensions associated with a weight matrix, the first two are number of feature maps of the convolutional input layer ( $F$ ) and number of feature

maps of the convolutional output later ( $F_p$ ). The other two are the size of the receptive field width and receptive field height.

In CNN a subset is convolved instead of the whole input data or image and the goal is to find similar patterns in the input data irrespective of the position of the pattern in the data. The pixel relationships are preserved by learning image features using small squares of input data. The size of the resulting output image (width x height) is determined by the stride. The stride is simply the number of pixel by which one slides in the vertical and/or the horizontal direction before applying again the convolution operation. Normally ReLU is used at the convolutional layers.

In short convolution is a mathematical operation that takes two inputs such as an image matrix and a filter. The convolutions in CNN can often become complex, but aids to resolve complex problems.

## **Pooling Layer**

The pooling layers are introduced to reduce the number of parameters, these are optional and used when the data or image is too large. You may apply subsampling to reduce the dimensionality of each map retaining valuable information. The subsampling is otherwise called as down sampling or spatial pooling.

There are different type of spatial pooling, they are given below.

- Max Pooling
- Average Pooling
- Sum Pooling

Max pooling choose the largest element from the receptive field. Average pooling takes average value of receptive field, and sum pooling takes sum of elements in the feature map.

### **Batch Normalization**

Batch normalization allows every layer in the CNN to perform a more independent learning. You may simply consider this as a layer used to normalize the output of the previous layer. The activations scale the input layer in normalization. Batch normalization can be used to produce a more accurate and efficient CNN models which avoids the problem of overfitting. This layer is normally placed after convolution and pooling layers.

### **Fully connected layer**

After the convolution and pooling layers the rest is similar to any feed forward networks. We flattened our matrix into vector and feed it into a fully connected layer like a neural network.

Consider the obtained feature map matrix is converted and turned to a vector  $X = (x_1, x_2, \dots, x_n)$ , this is fed to fully connected layers to create a model. We can apply any activation function at this stage to produce an output, the choice of activation function is the freedom of the model designer. you can choose activation functions such as softmax, sigmoid etc.

### 5.5.2. Types of CNN

There are a number of Convolutional neural networks and some of them are listed below.

- LeNet-5

LeNet is an exceptional CNN introduced to perform object classification, it is exceptional in its performance, accuracy and capability. This algorithm was originally developed and trained to classify 0-9 hand written digits from MNIST dataset. LeNet consists of 7 layers and the activation function applied is RELU.

- AlexNet

AlexNet, was introduced by Alex Krizhevsky, Ilya Sutskever and Geoffery E. Hinton and won ImageNet ILSVRC-2012 competition. This CNN is very similar to the LeNet structure, but consist of a large number of filters in contrast to original LeNet. AlexNet is able to classify large set of objects. To deal issues of overfitting, AlexNet make use of “dropout” instead of regularization to deal with overfitting.

- VGGNet-16

VGGNet-16 was designed by Simonyan and Zisserman and became the runners up of the ILSVRC-2014 competition. VGGNet is exceptional for feature extraction because of the simple pattern that it follows. The hyperparameters for the filter size and the strides for both of the convolution layer and the pooling layer are constant: Filters of size 3 X 3 and stride = 1 used at the convolution layer and the max pooling layer make



use of filters of size  $2 \times 2$  and stride = 2. These layers are applied in a particular order all over the network

- ResNet

ResNet provide better performance than VGGNet and came first in all the awards of the ILSVRC 2015 for classification, detection and localization. ResNet is otherwise called as Residual Networks and was the most ground breaking development happened in the area of CNN so far. ResNet make use of “skip-connections” and “ heavy batch-normalizations” to aid training with thousands of layers without performance degradation. Performance degradation means the problem of “vanishing gradient” where the back propagated gradient becomes infinitely small.

ResNet used the idea of “identity shortcut connection” that transfer results to few deeper layers and skips some other layers in between.

- GoogleNet/Inception

GoogleNet was developed by Google. GoogleNet or the Inception Network became the winner of ILSVRC 2014 competition and achieved a top-5 error rate of 6.67%. These type of error rate is very similar to human error rare. The model is a very innovative implementation of basic LeNet architecture. The model was developed by Google and includes a smarter implementation of the original LeNet architecture. Instead of implementing convolutional layers of various hyperparameters in different layers, all the convolutions are done together to output a result containing matrices from all the filter operations together.