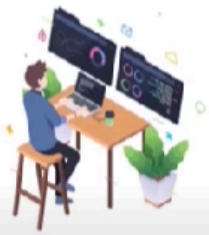
Machine Learning

The most important aspect of Data Science



- Forms to be an important application of Artificial Intelligence (AI)
- Programs learn from experience and improve on their own.
- The goal is to make machines more human-like.
- Where does it learn from?
  - **Datasets**

Why Machine Learning?

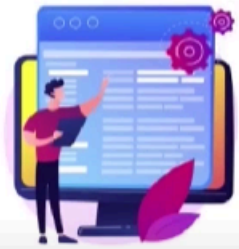
Lots of reasons!

- Helps reduce production cost
- Ability to easily process large amounts of data
- Deriving key insights about businesses
- Finding out hidden trends in data

Types of Machine Learning

Machine Learning

There are 3 types of Machine Learning:

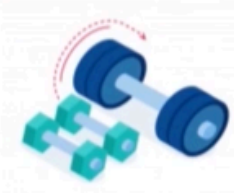


- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

Supervised Learning

- Presence of data and associated **labels** for the data

Flowers



Dumbbells

Car

Supervised Learning

- **y = f(x)** forms to be the foundation of supervised learning.
- The input variable is ‘x’, while the output variable is ‘y’.
- Mapping the output as a function of the input variable.

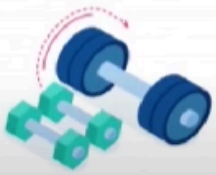
Supervised Learning

Grouping

- **Regression** – Prediction of future values from past data
- **Classification** – Categorization of items using data.

Unsupervised Learning

- Goal of the algorithm is to find a structure present in the data.
- Nothing apart from the input variables are present.


Entirely contrast to supervised learning
Based upon how the machine learning algorithm is going to find out the hidden patterns with the data we gave.
Complex and takes more computing time and power to find the patterns when compared to supervised learning.

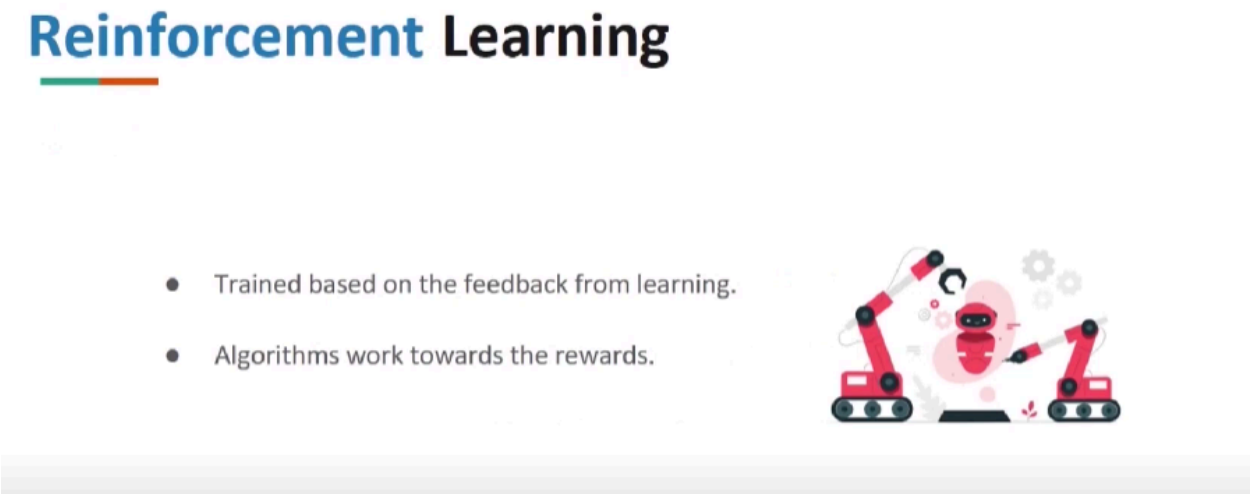
Unsupervised Learning

Grouping

- **Clustering** – Grouping of input variables with similar characteristics.
- **Association** – Mapping associations based on data (People will also watch..)

Clustering : grouping based on the category like separating the bunch of erasers and pencils into separate piles.
Association : we have been already using this…
We all are into online shopping. Let's say it's amazon shopping website
As soon as we buy something or we add an item into cart it automatically suggests **"People Also bought this"** and yes these recommender systems are machine learning techinques.
Ex 2:- Movie recommender systems used in netflix,amazon based on the user preferences and trending movies by showing **"Similar searches found".**

Book recommender systems



Trained based on the feedback from learning
Algorithm works towards the rewards

Dogs trained professionally
Give Handshake - get a biscuit as a reward
Sit,stand,catch,roll over
Same here - ML Algorithm learns based on the feedback we give to get rewards

Positive feedback = +1 (rewards)
Negative feedback = -1 (penalty)

ML Model is very different from ML Algorithm
Once we have data we create the algorithm and we use the algorithm along with the data to get the results.

After the algorithm is trained it is no longer gonna say algorithm anymore.
We call it a MODEL.

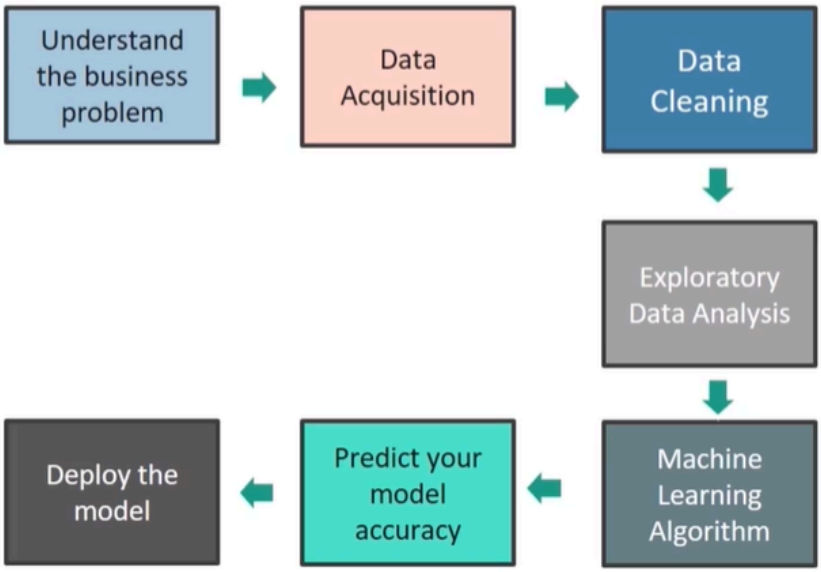
How does a Machine Learning model learn?

Data is split into two parts!

- **Training** Data – Used to teach the algorithm
- **Testing** Data – Used to verify the learning capability

70-30 / 80-20 training and testing data split usually.

How does a Machine Learning model learn



Apply the Machine algorithm once after preparing the data

Python for Machine Learning

Python for Machine Learning

Python is a powerful programming language!

- Python is the most popular programming language
- Python is the preferred choice for Machine Learning
- Easy to work with and high-level syntax

Python for Machine Learning

Installing and working with Python is a breeze!

- PyCharm IDE
- Python website
- Anaconda
- Google Colaboratory

Python for Machine Learning

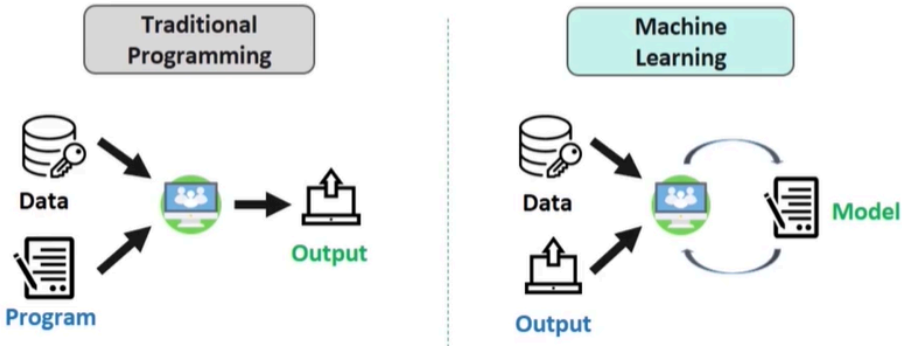
Data manipulation libraries

NumPy

Pandas

Traditional programming vs Machine Learning

Very important difference!



Machine Learning Algorithms

Machine Learning Algorithms

Many types of algorithms based on working!

- Linear Regression
- Logistic Regression
- Naïve Bayes
- K-NN Algorithm
- Support Vector Machines (SVM)
- Random Forests

1. Linear Regression

Linear Regression

- What is regression?
  - Modelling a target value based on independent variables.
- Why is it so popular?
  - Mainly used for finding out cause-effect relationship between variables.

Linear Regression

Let us implement a simple Linear Regression model!

- Boston housing data

Predicting the price of the houses

Linear Regression

Importing the dataset into a pandas DataFrame

```
[ ]  import numpy as np
     import pandas as pd
     import sklearn
     column_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX','PTRATIO', 'B', 'LSTAT', 'MEDV']
     bos1 = pd.read_csv('housing.csv', delimiter=r"\s+", names=column_names)
     bos1.head()
```

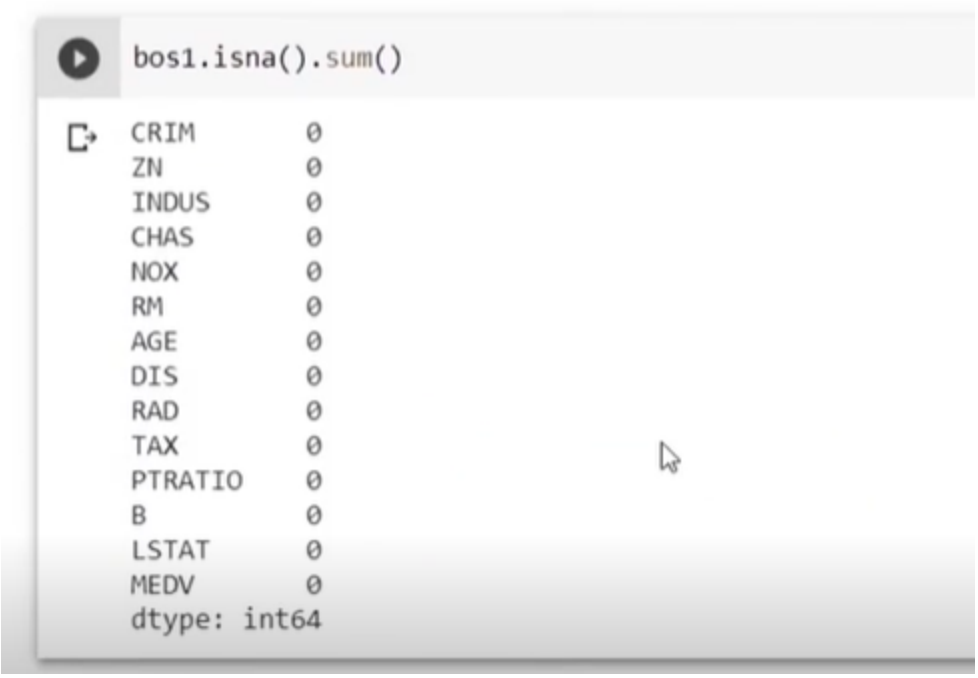
```
import numpy as np
import pandas as pd
import sklearn
column_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX','PTRATIO', 'B', 'LSTAT', 'MEDV']
bos1 = pd.read_csv('housing.csv', delimiter=r"\s+", names=column_names)
bos1.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

```
bos1 = pd.read_csv('housing.csv', delimiter=r"\s+", names=column_names)
bos1.head(10)
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2
5	0.02985	0.0	2.18	0	0.458	6.430	58.7	6.0622	3	222.0	18.7	394.12	5.21	28.7
6	0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311.0	15.2	395.60	12.43	22.9
7	0.14455	12.5	7.87	0	0.524	6.172	96.1	5.9505	5	311.0	15.2	396.90	19.15	27.1
8	0.21124	12.5	7.87	0	0.524	5.631	100.0	6.0821	5	311.0	15.2	386.63	29.93	16.5
9	0.17004	12.5	7.87	0	0.524	6.004	85.9	6.5921	5	311.0	15.2	386.71	17.10	18.9

Preprocessing the data: Removing NaN values



```
bos1.isna().sum()

CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```

`df.isna()`: This function is used to create a Boolean DataFrame of the same shape as the original DataFrame `df`, where each cell contains `True` if the corresponding cell in the original DataFrame is missing (NaN or None), and `False` otherwise.
`df.isna().sum()`: This function is applied on the Boolean DataFrame returned by `df.isna()`. It calculates the sum of `True` values (missing data) for each column in the DataFrame. Since `True` is considered as 1 and `False` is considered as 0 in numerical calculations, summing up the `True` values effectively gives you the count of missing values in each column.

In summary, by using `df.isna().sum()`, you can obtain a count of missing values for each column in your DataFrame. This information is crucial for understanding the quality and completeness of your data, as well as for making decisions about how to handle missing values during the data preprocessing and analysis stages.

We can observe our data is properly preprocessed → No Missing values in each column

Splitting model data with 70% for training.

```
from sklearn.model_selection import train_test_split
X=np.array(bos1.iloc[:,0:13])
Y=np.array(bos1["MEDV"])
#testing data size is of 30% of entire data
x_train, x_test, y_train, y_test =train_test_split(X,Y, test_size = 0.30, random_state =5)
```

Train and test data splitting

From sklearn.model_selection import train_test_split
x=np.array(df.iloc[:,0:13])
y=np.array(df[“MEDV”])
x_train,x_test,y_train,y_test =
train_test_split(x,y,test_size=0.30,random_state=5)

Explanation:

1. `from **sklearn.model_selection** import **train_test_split**`: This line imports the `**train_test_split**` function from the `s**klearn.model_selection**` module. This function is used to split a dataset into training and testing subsets, which is a crucial step in building and evaluating machine learning models.

2. `x = np.array(df.iloc[:, 0:13])`: This line extracts the input features (independent variables) from the DataFrame `df`. It uses the `.iloc` method to index and select rows and columns from the DataFrame. Here, `[:, 0:13]` selects all rows and the columns indexed from 0 to 12 (a total of 13 columns). The extracted features are stored in the NumPy array `x`.

3. `y = np.array(df["MEDV"])`: This line extracts the target variable (dependent variable) from the DataFrame `df`. It selects the column with the label `"MEDV"` and stores the values in the NumPy array `y`.

4. `x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30, random_state=5)`: This line uses the `train_test_split` function to split the data into training and testing subsets. Here's what the arguments mean:

- `x`: The input features (independent variables).
- `y`: The target variable (dependent variable).

- `**test_size**`: This parameter specifies the proportion of the data that should be allocated to the testing set. In this case, it's set to `0.30`, which means 30% of the data will be used for testing.

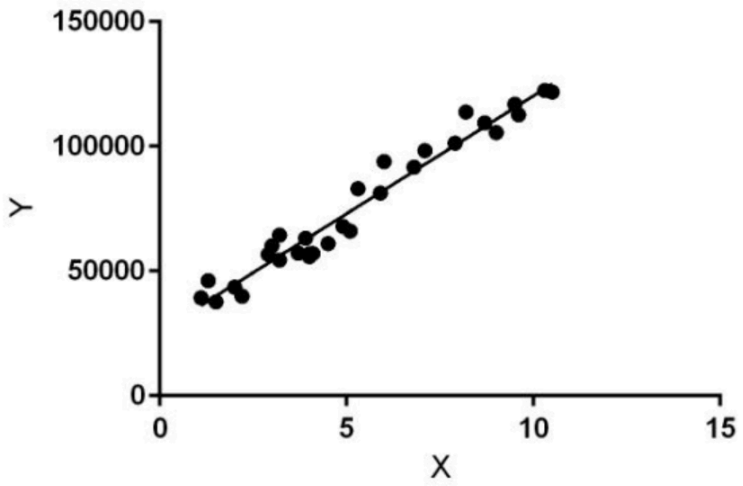
- `**random_state**`: This parameter sets the random seed for reproducibility. By using the same `random_state`, you ensure that the data split will be the same every time you run the code. This is useful for getting consistent results during development and testing.

The function returns four arrays:
- `x_train`: The training set of input features.
- `x_test`: The testing set of input features.
- `y_train`: The training set of target variable values.
- `y_test`: The testing set of target variable values.

Linear Regression Introduction

Linear Regression

Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of **relationship between dependent and independent variables** they are considering and the number of independent variables being used.



Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression.
In the figure above, X (input) is the work experience and Y (output) is the salary of a person. The regression line is the best fit line for our model.

Hypothesis function for Linear Regression :

$$y = \theta_1 + \theta_2.x$$

While training the model we are given :

x: input training data (univariate – one input variable(parameter))

y: labels to data (supervised learning)

If the data is not univariate, then we take the dot product of the theta(one dimensional matrix of randomized values which update as the minimisation function proceeds) and the X variable(the entire dataset)

When training the model – it fits the best line to predict the value of y for a given value of x. The model gets the best regression fit line by finding the best θ1 and θ2 values.

θ1: intercept

θ2: coefficient of x

Once we find the best θ1 and θ2 values, we get the best fit line. So when we are finally using our model for prediction, it will predict the value of y for the input value of x.

How to update θ1 and θ2 values to get the best fit line ?

Cost Function (J):

By achieving the best-fit regression line, the model aims to predict y value such that the error difference between predicted value and true value is minimum. So, it is very important to update the θ1 and θ2 values, to reach the best value that minimizes the error between predicted y value (pred) and true y value (y).

$$minimize \frac{1}{n} \sum_{i=1}^{n} (pred_i - y_i)^2$$

$$J = \frac{1}{n} \sum_{i=1}^{n} (pred_i - y_i)^2$$

Cost function(J) of Linear Regression is the Root Mean Squared Error (RMSE) between predicted y value (pred) and true y value (y).

Gradient Descent:

To update θ1 and θ2 values in order to reduce Cost function (minimizing RMSE value) and achieve the best fit line the model uses Gradient Descent. The idea is to start with random θ1 and θ2 values and then iteratively update the values, reaching minimum cost.

Cost function → RMSE Between PredictedYb(pred) value and TrueY value (y)

Gradient Descent in Linear Regression

In linear regression, the model targets to get the best-fit regression line to predict the value of y based on the given input value (x). While training the model, the model calculates the cost function which measures the Root Mean Squared error between the predicted value (pred) and true value (y). The model targets to minimize the cost function.

To minimize the cost function, the model needs to have the best value of θ_1 and θ_2. Initially the model selects θ_1 and θ_2 values randomly and then iteratively update these values in order to minimize the cost function until it reaches the minimum. By the time the model achieves the minimum cost function, it will have the best θ_1 and θ_2 values. Using these finally updated values of θ_1 and θ_2 in the hypothesis of linear equation, the

model predicts the value of x in the best manner it can.

Therefore, the question arises – How do θ_1 and θ_2 values get updated?

$$J = \frac{1}{n}\sum_{i=1}^{n}(pred_i - y_i)^2$$

Linear Regression Cost Function:

$$minimize\frac{1}{n}\sum_{i=1}^{n}(pred_i - y_i)^2$$

Gradient Descent Algorithm For Linear Regression

Using Linear Regression Model

```
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
#load our first model
lr = LinearRegression()
#train the model on training data
lr.fit(x_train,y_train)
#predict the testing data so that we can later evaluate the model
pred_lr = lr.predict(x_test)
```

-> θ_j : Weights of the hypothesis.

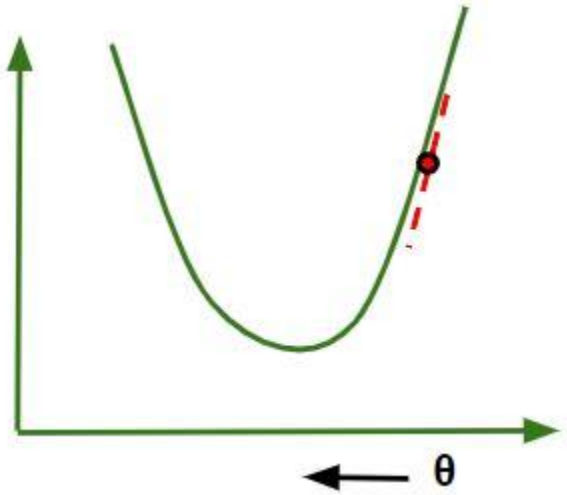
-> $h_{\theta(xi)}$: predicted y value for i^{th} input.

-> j : Feature index number (can be 0, 1, 2, ..., n).

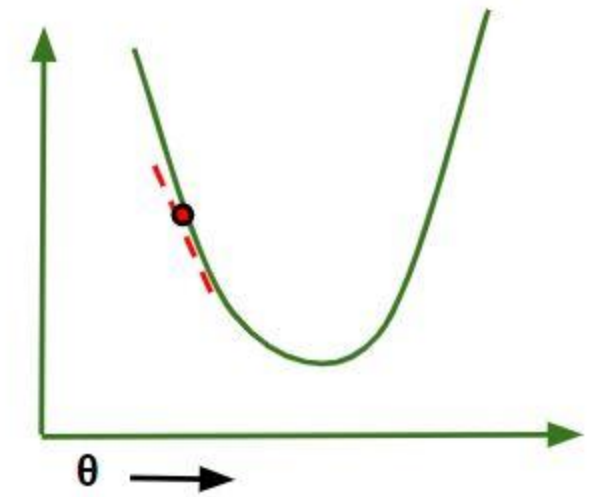
-> α : Learning Rate of Gradient Descent

We graph cost function as a function of parameter estimates i.e. parameter range of our hypothesis function and the cost resulting from selecting a particular set of parameters. We move downward towards pits in the graph, to find the minimum value. The way to do this is taking derivative of cost function as explained in the above figure. Gradient Descent step-downs the cost function in the direction of the steepest descent. The size of each step is determined by parameter α known as Learning Rate.
In the Gradient Descent algorithm, one can infer two points :

- If slope is +ve : $\theta_j = \theta_j$ – (+ve value). Hence value of θ_j decreases.

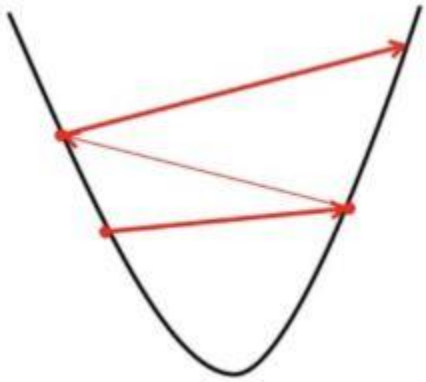


- If slope is -ve : $\theta_j = \theta_j$ – (-ve value). Hence value of θ_j increases.



The choice of correct learning rate is very important as it ensures that Gradient Descent converges in a reasonable time. :

- If we choose α to be very large, Gradient Descent can overshoot the minimum. It may fail to converge or even diverge.



- If we choose α to be very small, Gradient Descent will take small steps to reach local minima and will take a longer time to reach minima.

For linear regression Cost, the Function graph is always convex shaped

Updating the Parameters in Linear Regression

Steps to update Parameter in Linear Regression

Step 1:

Forward Propagation :

let the hypothesis function be y=mx+c

Step 2:

Differentiation of Cost Function with respect to slope and constant :

To find out the best fit line we have to minimize the cost function, we differentiate the cost function with respect to slope and constant "J"

Multiple Linear Regression Intuition

Now that you are reminded what simple linear regression, we can move onto multiple linear regression. MLR is same thing but with more than one input variables. Here it is how it looks in mathematical equation.

$$y = w1x1 + w2x2 + w3x3 + b$$

Depending on where you look, all variables can have different names, but I'll try to keep it simple with other commonly used terms.

y — value we want to predict/dependent variable/predicted value

Xi — features / independent variable / expanatory variable / observed variable

Wi — coefficient for feature

b- bias /constant value

So simplified, we are predicting what value of y will be depending on features Xi and with coefficients Wi we are deciding how much each feature is affecting predicted value.

$$y = b + \sum (WiXi)$$

Why is it called linear regression?

There are multiple features, but all coefficients and features in equation are linear. No variable has exponential higher than one. And that is why it is called linear. Otherwise we would have polynomial regression.

Gradient Descent

$$w1' = w1 - \alpha * \frac{\partial J}{\partial w1}$$
$$w2' = w2 - \alpha * \frac{\partial J}{\partial w2}$$
$$w3' = w3 - \alpha * \frac{\partial J}{\partial w3}$$
$$c' = c - \alpha * \frac{\partial J}{\partial c}$$
$$\frac{\partial J}{\partial w1} = \frac{1}{n} * (\hat{y} - y) * x1$$

$$\frac{\partial J}{\partial w2} = \frac{1}{n} * (\hat{y} - y) * x2$$

$$\frac{\partial J}{\partial w3} = \frac{1}{n} * (\hat{y} - y) * x3$$

$$\hat{y} = mx + c$$

$$cost(J) = \frac{1}{2n}(\hat{y} - y)^2$$

$$m' = m - \alpha \frac{\partial cost}{\partial m}$$

$$c' = c - \alpha \frac{\partial cost}{\partial c}$$

$$\frac{\partial cost}{\partial m} = \frac{\partial}{\partial m}\left(\frac{1}{2n}(\hat{y} - y)^2\right)$$

$$\frac{\partial cost}{\partial m} = \frac{2}{2n}(\hat{y} - y)\frac{\partial}{\partial m}(mx + c - y)$$

$$= \frac{2}{2n}(\hat{y} - y)\, x$$

$$\frac{\partial cost}{\partial m} = \frac{(\hat{y} - y)x}{n}$$

$$\frac{\partial cost}{\partial c} = \frac{2}{2n}(\hat{y}-y)\frac{\partial}{\partial c}(mx+c-y)$$

$$= \frac{1}{n}(\hat{y}-y)*1$$

$$\frac{\partial cost}{\partial c} = \frac{(\hat{y}-y)}{n}$$

$$\therefore \quad m' = m - \alpha\left(\frac{\hat{y}-y}{n}\right)x$$

$$c' = c - \alpha\frac{(\hat{y}-y)}{n}$$

Linear Regression Model Assumption

In the Linear Regression algorithm you have to take some assumption as it is a parametric approach which means it gives wrong output on some dataset which will not fulfill its condition's ,

Assumptions of Multiple Linear Regression:

Simple Linear Regression:

(1) 

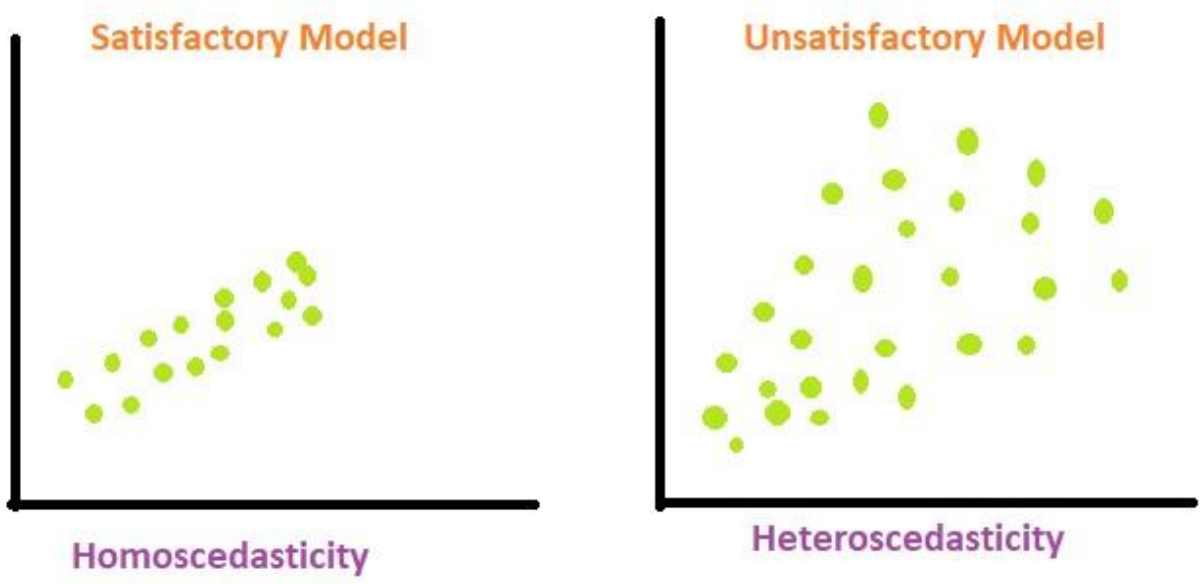
(2) 

should be always be linear of degree '1' else it gives poor result.

1. Error has zero meane
2. Error has constant variance
3. Errors are uncorrelated
4. Errors are normally distributed

The *second assumption* is known as Homoscedasticity and therefore, the violation of this assumption is known as Heteroscedasticity.

Homoscedasticity *vs* Heteroscedasticity:



Therefore, in simple terms, we can define heteroscedasticity as the condition in which the variance of error term or the residual term in a regression model varies. As you can see in the above diagram, in case of homoscedasticity, the data points are equally scattered while in case of heteroscedasticity the data points are not equally scattered.

Possible reasons of arising Heteroscedasticity:

1. Often occurs in those data sets which have a large range between the largest and the smallest observed values i.e. when there are outliers.
2. When model is not correctly specified.
3. If observations are mixed with different measures of scale.
4. When incorrect transformation of data is used to perform the regression.
5. Skewness in the distribution of a regressor, and may be some other sources.

Effects of Heteroscedasticity:

- As mentioned above that one of the assumption (assumption number 2) of linear regression is that there is no heteroscedasticity. Breaking this assumption means that *OLS (Ordinary Least Square)* estimators are not the *Best Linear Unbiased Estimator(BLUE)* and their variance is not the lowest of all other unbiased estimators.
- Estimators are no longer *best/efficient.*
- The tests of hypothesis (like t-test, F-test) are no longer valid due to the inconsistency in the co-variance matrix of the estimated regression coefficients.

Identifying Heteroscedasticity with residual plots:
As shown in the above figure, heteroscedasticity produces either outward opening funnel or outward closing funnel shape in residual plots.

Identifying Heteroscedasticity Through Statistical Tests:
The presence of heteroscedasticity can also be quantified using the algorithmic approach. There are some statistical tests or methods through which the presence or absence of heteroscedasticity can be established.

1. The Breush – Pegan Test: It tests whether the variance of the errors from regression is dependent on the values of the independent variables. In that case, heteroskedasticity is present.
2. White test: White test establishes whether the variance of the errors in a regression model is constant. To test for constant variance one undertakes an auxiliary regression analysis: this regresses the squared residuals from the

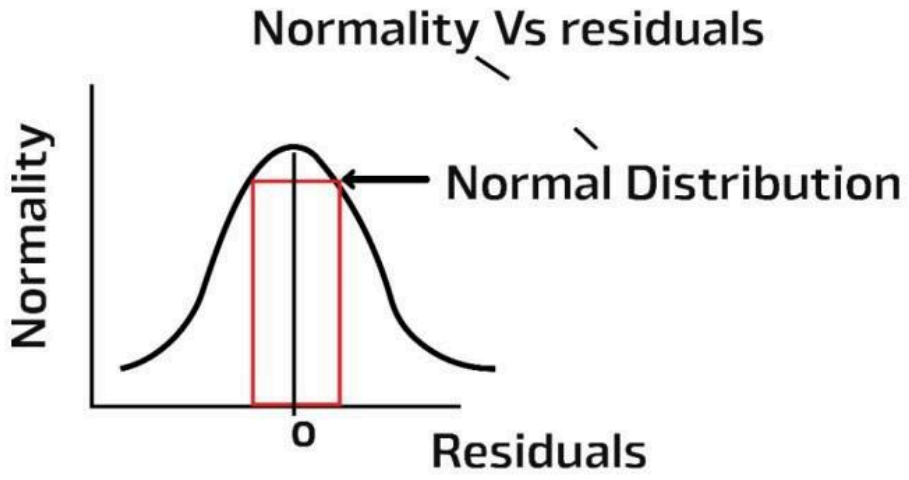
original regression model onto a set of regressors that contain the original regressors along with their squares and cross-products.

Corrections for heteroscedasticity:

1. We can use different specification for the model.
2. Weighted Least Squares method is one of the common statistical method. This is the generalization of ordinary least square and linear regression in which the errors co-variance matrix is allowed to be different from an identity matrix.
3. Use MINQUE: The theory of *Minimum Norm Quadratic Unbiased Estimation (MINQUE)* involves three stages. First, defining a general class of potential estimators as quadratic functions of the observed data, where the estimators relate to a vector of model parameters. Secondly, specifying certain constraints on the desired properties of the estimators, such as unbiasedness and third, choosing the optimal estimator by minimizing a “norm” which measures the size of the covariance matrix of the estimators.

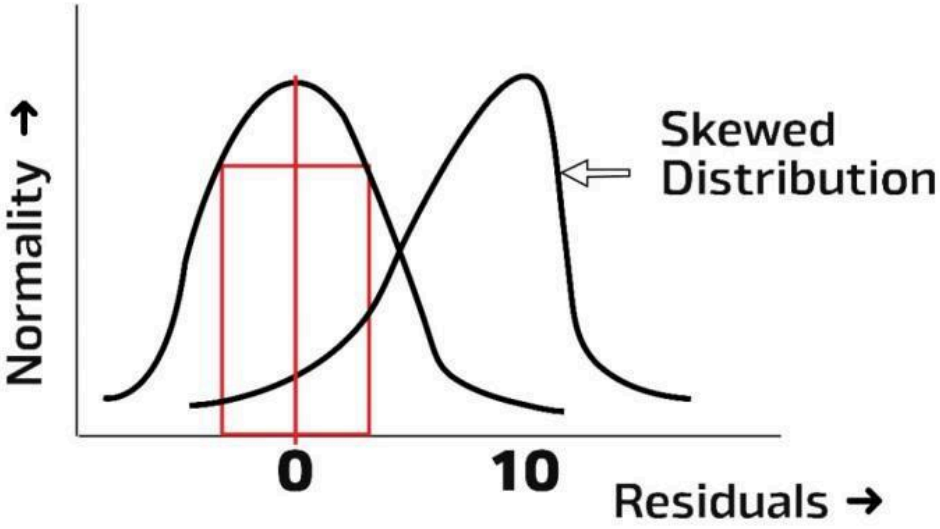
Normality Distribution:

The data should in distributed normally such that the curve formed is bell shaped, means most of the errors in standard deviation graph should lies close to zero meanwhile we want error to be minimum as much as possible.



Skewed Normal Distribution

when the data distributed in such a way that the curve tilt towards right and error increases significantly is called skewed distribution



Polynomial Linear Regression Intuition

Polynomial Regression is a form of linear regression in which the relationship between the independent variable x and dependent variable y is modeled as an *nth* degree polynomial. Polynomial regression fits a nonlinear relationship between the value of x and the corresponding conditional mean of y, denoted E(y |x)
Why Polynomial Regression:

- There are some relationships that a researcher will hypothesize is curvilinear. Clearly, such types of cases will include a polynomial term.
- Inspection of residuals. If we try to fit a linear model to curved data, a scatter plot of residuals (Y-axis) on the predictor (X-axis) will have patches of many positive residuals in the middle. Hence in such a situation, it is not appropriate.
- An assumption in usual multiple linear regression analysis is that all the independent variables are independent. In polynomial regression model, this assumption is not satisfied.

Uses of Polynomial Regression:
These are basically used to define or describe non-linear phenomena such as:

- The growth rate of tissues.
- Progression of disease epidemics
- Distribution of carbon isotopes in lake sediments

The basic goal of regression analysis is to model the expected value of a dependent variable y in terms of the value of an independent variable x. In simple regression, we used the following equation –

$y = a + bx + e$

Here y is a dependent variable, a is the y-intercept, b is the slope and e is the error rate. In many cases, this linear model will not work out For example if we analyzing the production of chemical synthesis in terms of temperature at which the synthesis take

place in such cases we use a quadratic model

y = a + b1x + b2^2 + e

Here y is the dependent variable on x, a is the y-intercept and e is the error rate.
In general, we can model it for nth value.

y = a + b1x + b2x^2 +....+ bnx^n

Since regression function is linear in terms of unknown variables, hence these models are linear from the point of estimation.
Hence through the Least Square technique, let's compute the response value that is y.
Polynomial Regression in Python:
To get the Dataset used for the analysis of Polynomial Regression,

Step 1: Import libraries and dataset
Import the important libraries and the dataset we are using to perform Polynomial Regression.

Python3

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
datas = pd.read_csv('data.csv')

datas
```

```
Out[3]:
   sno  Temperature  Pressure
0    1            0    0.0002
1    2           20    0.0012
2    3           40    0.0060
3    4           60    0.0300
4    5           80    0.0900
5    6          100    0.2700
```

Step 2: Dividing the dataset into 2 components

Divide dataset into two components that is X and y.X will contain the Column between 1 and 2. y will contain the 2 columns.

Python3

```
X = datas.iloc[:, 1:2].values
y = datas.iloc[:, 2].values
```

Step 3: Fitting Linear Regression to the dataset

Fitting the linear Regression model On two components.

Python3

```
# Fitting Linear Regression to the dataset
from sklearn.linear_model import LinearRegression
lin = LinearRegression()

lin.fit(X, y)
```

Step 4: Fitting Polynomial Regression to the dataset

Fitting the Polynomial Regression model on two components X and y.

Python3

```
# Fitting Polynomial Regression to the dataset
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree = 4)
```

```
X_poly = poly.fit_transform(X)

poly.fit(X_poly, y)
lin2 = LinearRegression()
lin2.fit(X_poly, y)
```

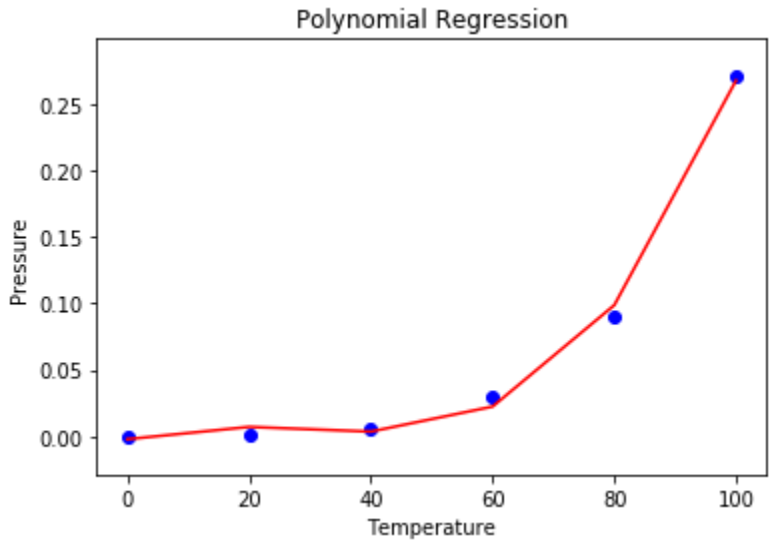
Step 5: In this step, we are Visualising the Linear Regression results using a scatter plot.

Python3

```
# Visualizing the Linear Regression results
plt.scatter(X, y, color = 'blue')

plt.plot(X, lin.predict(X), color = 'red')
plt.title('Linear Regression')
plt.xlabel('Temperature')
plt.ylabel('Pressure')

plt.show()
```



Advantages of using Polynomial Regression:

- A broad range of functions can be fit under it.
- Polynomial basically fits a wide range of curvatures.
- Polynomial provides the best approximation of the relationship between dependent and independent variables.

Disadvantages of using Polynomial Regression

These are too sensitive to the outliers.

The presence of one or two outliers in the data can seriously affect the results of nonlinear analysis.

In addition, there are unfortunately fewer model validation tools for the detection of outliers in nonlinear regression than there are for linear regression.

2. Naïve Bayes Algorithm

Naïve Bayes Algorithm

- Based on Bayes theorem
- Supervised learning
- Called "**Naïve**" because of no co-dependency in attributes.
- Very popular social media marketing and product management.

Naive bayes is based on the Bayes Theorem - Works on the conditional probability
Conditional probability is defined as the likelihood of an event or outcome occurring, based on the occurrence of a previous event or outcome.

Naïve Bayes Algorithm

Let us implement a simple Naïve Bayes model!

- Social Network Dataset

Naive Bayes Algorithm

+ Code + Text

Bringing in all the libraries

```
import pandas as pd
import numpy as np
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
```

Importing training and testing dataset

Bringing in all the libraries

```
import pandas as pd
import numpy as np
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
```

Importing training and testing dataset


```
[ ] salary_train = pd.read_csv("SalaryData_Train.csv")
    salary_test = pd.read_csv("SalaryData_Test.csv")
    string_columns=["workclass","education","maritalstatus","occupation","relationship","race","sex","native"]
    salary_train.head()
```

	age	workclass	education	educationno	maritalstatus	occupation	relationship	race	sex	capitalgain	capitalloss	ho
0	39	State-gov	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	
1	50	Self-emp-not-inc	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	
2	38	Private	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	
3	53	Private	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	
4	28	Private	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female			

educationno	maritalstatus	occupation	relationship	race	sex	capitalgain	capitalloss	hoursperweek	native	Salary
13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0			

Using a Label Encoder - Categorical to numerical

```
from sklearn import preprocessing
number = preprocessing.LabelEncoder()
for i in string_columns:
    salary_train[i] = number.fit_transform(salary_train[i])
    salary_test[i] = number.fit_transform(salary_test[i])
```

Label Encoder - converting categorical data into numerical data

Assign the encoded variables for training and testing

```
colnames = salary_train.columns
len(colnames[0:13])
trainX = salary_train[colnames[0:13]]
trainY = salary_train[colnames[13]]
testX = salary_test[colnames[0:13]]
testY = salary_test[colnames[13]]
```

Gaussian Naive Bayes classifier for model fitting

```
[ ]  sgnb = GaussianNB ()
```


```
colnames = salary_train.columns
len(colnames[0:13])
trainX = salary_train[colnames[0:13]]
trainY = salary_train[colnames[13]]
testX = salary_test[colnames[0:13]]
testY = salary_test[colnames[13]]
```


Gaussian Naive Bayes classifier for model fitting

```
sgnb = GaussianNB ()
```

Predictions for test data

```
[ ]  spred_gnb = sgnb.fit(trainX,trainY).predict(testX)
```

Predictions for test data

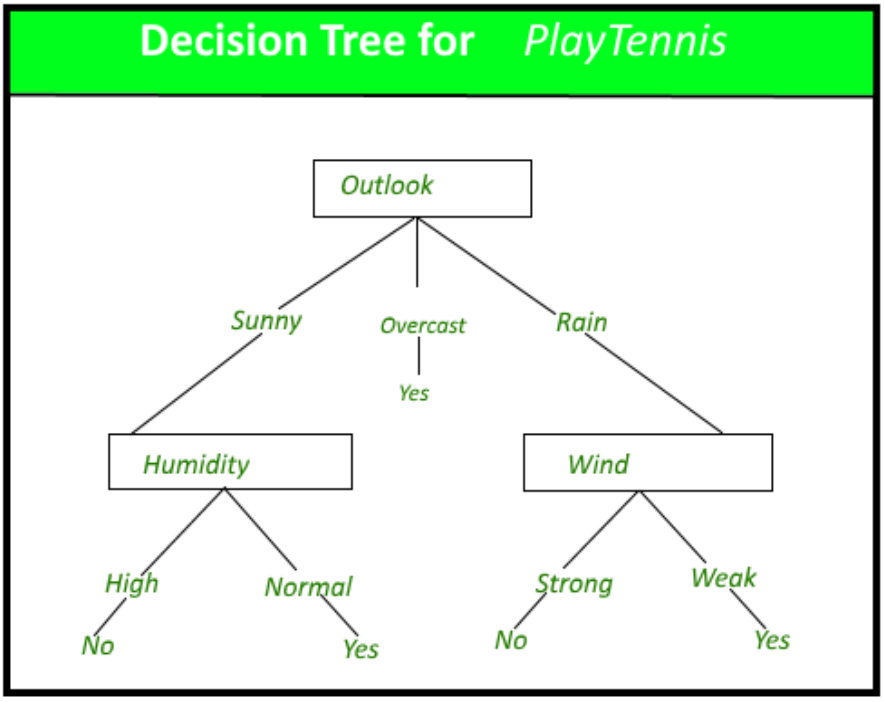
```
spred_gnb = sgnb.fit(trainX,trainY).predict(testX)
```

Confusion Matrix

```
from sklearn.metrics import confusion_matrix
cm_gnb = confusion_matrix(testY,spred_gnb)
print(cm_gnb)
print ("Accuracy", (10759+1209)/(10759+601+2491+1209))
```

Decision Tree Regression Intuition

Decision Tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart-like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.



A decision tree for the concept PlayTennis.

Construction of Decision Tree: A tree can be *"learned"* by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called *recursive partitioning*. The recursion is completed when the subset at a node all has the same value of the target variable, or when splitting no longer adds value to the predictions. The construction of a decision tree classifier does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery. Decision trees can handle high-dimensional data. In general decision tree classifier has good accuracy. Decision tree induction is a typical inductive approach to learn knowledge on classification.

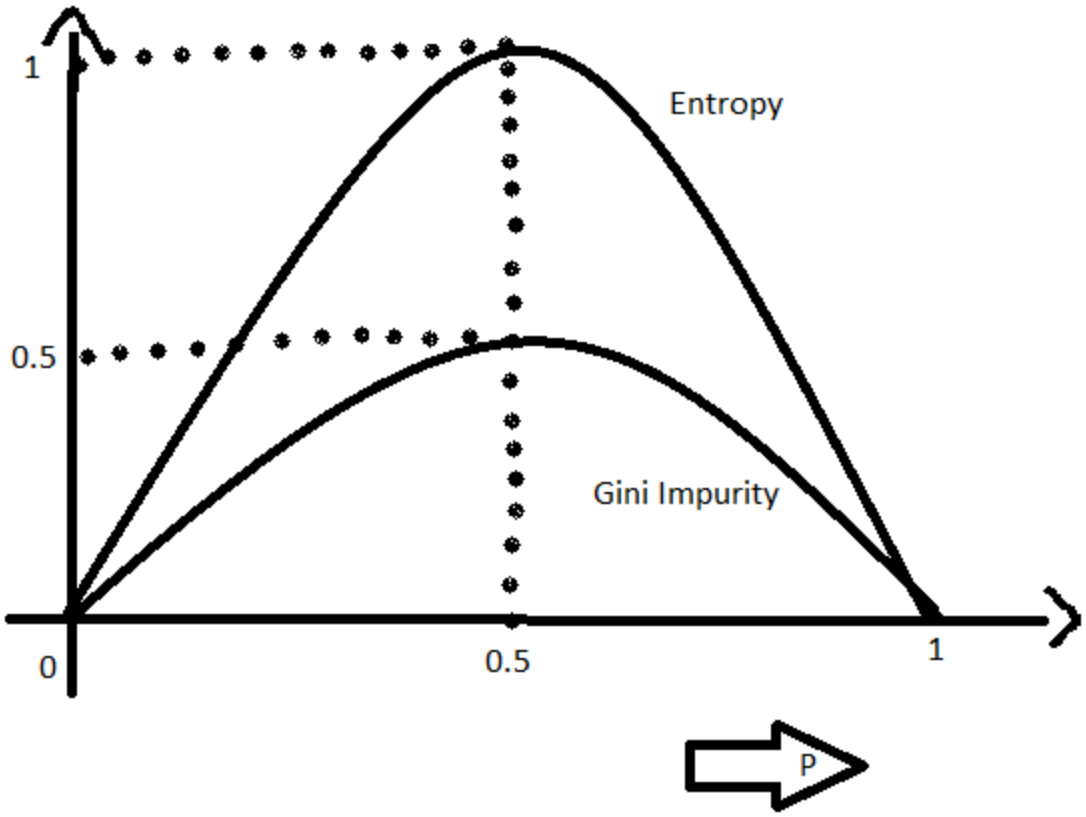
Entropy:

As discussed above entropy helps us to build an appropriate decision tree for selecting the best splitter. Entropy can be defined as a measure of the purity of the sub split. Entropy always lies between 0 to 1. The entropy of any split can be calculated by this formula.

The algorithm calculates the entropy of each feature after every split and as the splitting continues on, it selects the best feature and starts splitting according to it.

Gini Impurity:

The internal working of Gini impurity is also somewhat similar to the working of entropy in the Decision Tree. In the Decision Tree algorithm, both are used for building the tree by splitting as per the appropriate features but there is quite a difference in the computation of both the methods. Gini Impurity of features after splitting can be calculated by using this formula.



$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

$$Gini(E) = 1 - \sum_{j=1}^{c} p_j^2$$

Information gain:

Information gain is the entropy difference of target and information 1 with respect to target.

$$Gain(T, X) = Entropy(T) - Entropy(T, X)$$

G(PlayGolf, Outlook) = **E**(PlayGolf) – **E**(PlayGolf, Outlook)

= 0.940 – 0.693 = 0.247

Let's Take an example to understand the split in decision tree

Attributes **Classes**

Outlook	Temperature	Humidity	Windy	Play Golf
Rainy	Hot	High	FALSE	No
Rainy	Hot	High	TRUE	No
Overcast	Hot	High	FALSE	Yes
Sunny	Mild	High	FALSE	Yes
Sunny	Cool	Normal	FALSE	Yes
Sunny	Cool	Normal	TRUE	No
Overcast	Cool	Normal	TRUE	Yes
Rainy	Mild	High	FALSE	No
Rainy	Cool	Normal	FALSE	Yes
Sunny	Mild	Normal	FALSE	Yes
Rainy	Mild	Normal	TRUE	Yes
Overcast	Mild	High	TRUE	Yes
Overcast	Hot	Normal	FALSE	Yes
Sunny	Mild	High	TRUE	No

In the above dataset of golf we have Outlook,Temperature,Humidity,Wind as Input parameters and playing golf is output parameter, to spit the data we must have the knowledge of entropy of parent node.

Now that we know these parameters, we can start the construction of the decision tree. First, we need to determine the root node of the decision tree. As the dataset is split into two subtypes — Attributes and Class, we calculate the entropy for both, and the following Entropies are obtained.

E(Play Golf)

E(Play Golf, Outlook)

E(Play Golf, Temperature)

E(Play Golf, Humidity)

E(Play Golf, Windy)

After the calculation of the Entropies, we calculate the Information Gain.

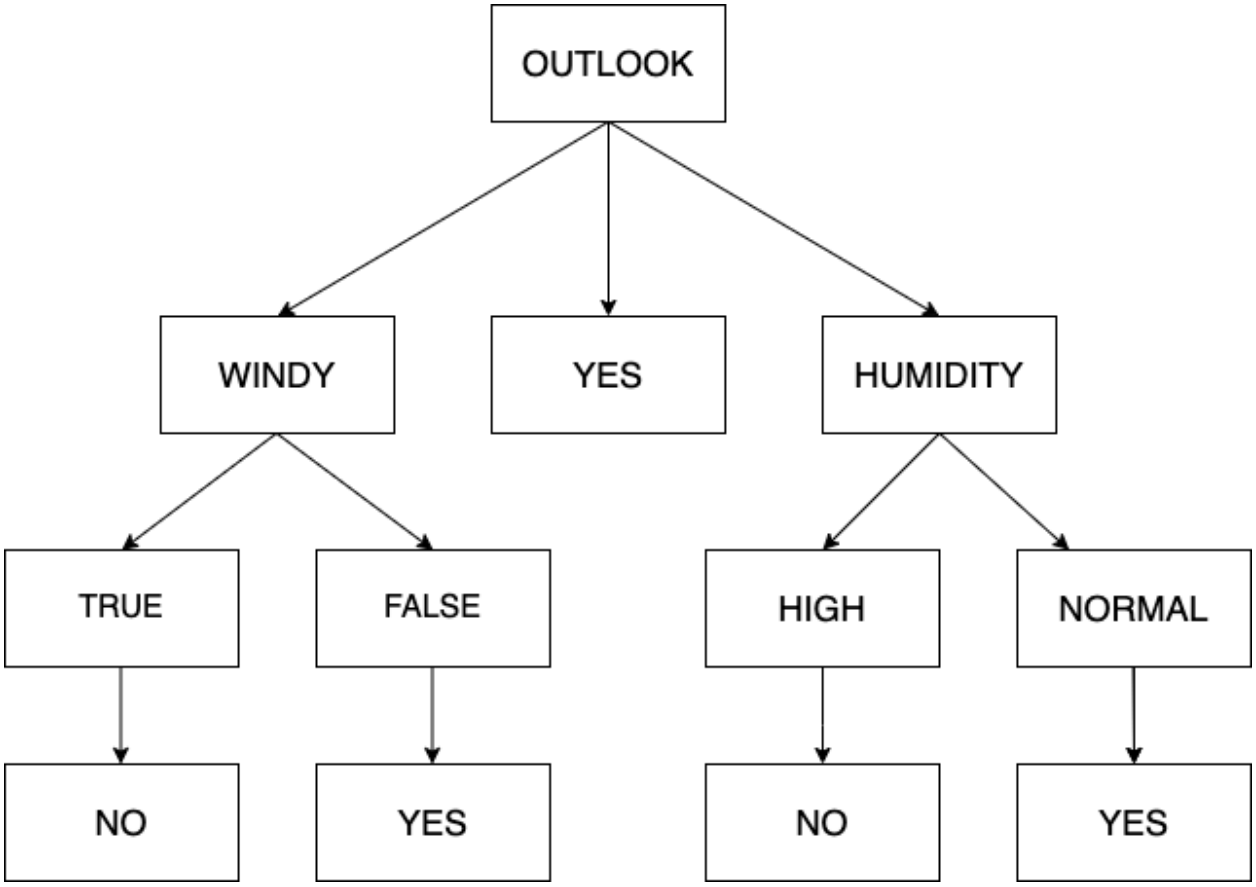
Gain(PlayGolf, Outlook) = Entropy(PlayGolf) — Entropy(PlayGolf, Outlook)

Gain(PlayGolf, Temperature) = Entropy(PlayGolf) — Entropy(PlayGolf, Temparature)

Gain(PlayGolf, Humidity) = Entropy(PlayGolf) — Entropy(PlayGolf, Humidity)

Gain(PlayGolf, Windy) = Entropy(PlayGolf) — Entropy(PlayGolf, Windy)

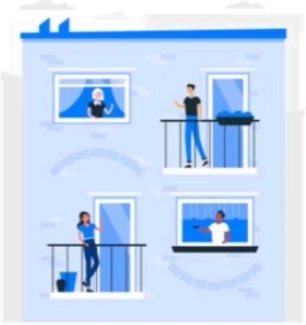
Now that we have all the necessary values, we can start the splitting. The first split i.e the root node is decided on the attribute which gives us the highest information gain. In this case, it is the Outlook attribute. The further splits will be decided based on which attribute gives us the homogeneous groups. The complete decision tree is shown below.



3. K-NN Algorithm

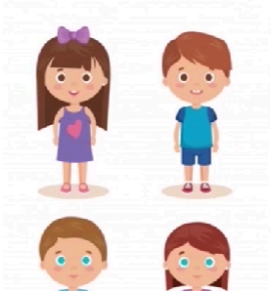
K-Nearest Neighbour Algorithm

- Input data is indexed to find the closest neighbors.
- Data is compared in the inferencing phase to save time.
- Belongs to the category of lazy learners!



K-Nearest Neighbour Algorithm

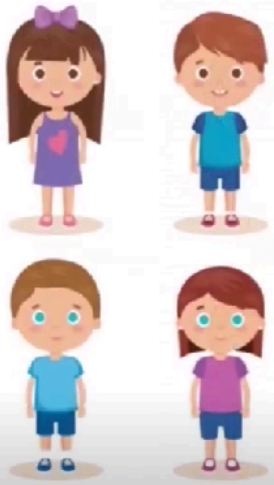
Let us understand a simple K-NN Classifier:



- Predict if person is male or female using K-NN.
- Prediction is based on height and weight of the person.

K-Nearest Neighbour Algorithm

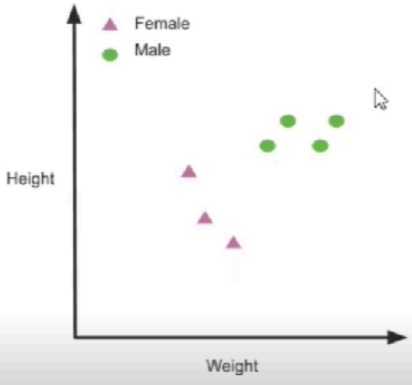
Let us understand a simple K-NN Classifier:



Height	Weight	Gender
187	80	Male
165	50	Female
199	99	Male
145	70	Female
180	87	Male
178	65	Female
187	60	Male

K-Nearest Neighbour Algorithm

Let us understand a simple K-NN Classifier:

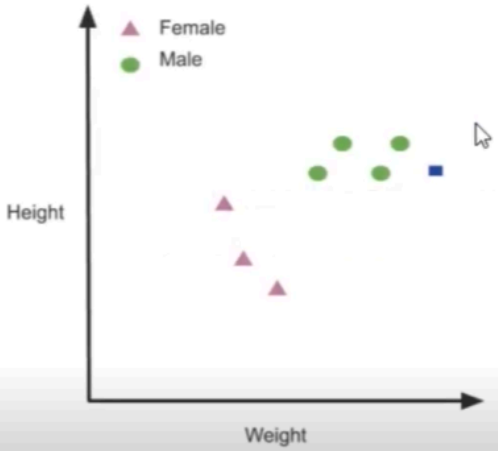


Now, need to add a new entry of a person with height 190cm and weight 100kg.

K-Nearest Neighbour Algorithm

Let us understand a simple K-NN Classifier:

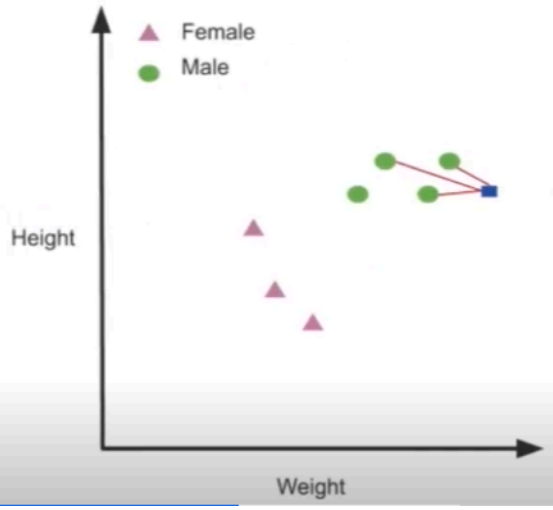
Added a new entry and plotted the relevant graph for it.



K-Nearest Neighbour Algorithm

Let us understand a simple K-NN Classifier:

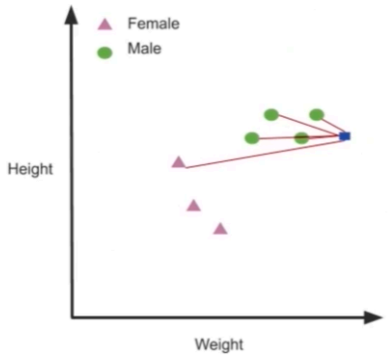
Assume k = 3:



K-Nearest Neighbour Algorithm

Let us understand a simple K-NN Classifier:

Assume k = 5:

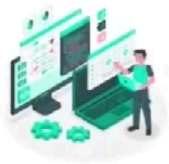


Note: Common practice to pick 'k' as an odd number if there are two classes in comparison.

K-Nearest Neighbour Algorithm

Practically checking out a simple K-NN Classifier:

- Boston housing data



K-NN Algorithm

Importing the dataset into a pandas DataFrame

```
import numpy as np
import pandas as pd
import sklearn
column_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX','PTRATIO', 'B', 'LSTAT', 'MEDV']
bos1 = pd.read_csv('housing.csv', delimiter=r"\s+", names=column_names)
bos1.head()
```

Splitting model data with 70% for training.

```
from sklearn.model_selection import train_test_split
X=np.array(bos1.iloc[:,0:13])
Y=np.array(bos1["MEDV"])
#testing data size is of 30% of entire data
x_train, x_test, y_train, y_test =train_test_split(X,Y, test_size = 0.30, random_state =5)
```

Using KNN Algorithm

```
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
#train the model on training data
lr.fit(x_train,y_train)
#load the KNN Model
Nn=KNeighborsRegressor(3)
Nn.fit(x_train,y_train)
pred_Nn = Nn.predict(x_test)
```

How does KNN work with examples?

It classifies the data point on how its neighbor is classified. **KNN classifies the new data points based on the similarity measure of the earlier stored data points**. For example, if we have a dataset of tomatoes and bananas. KNN will store similar measures like shape and color.

Hyperparameter Tuning

```
import sklearn
for i in range(1,50):
    model=KNeighborsRegressor(i)
    model.fit(x_train,y_train)
    pred_y = model.predict(x_test)
    mse = sklearn.metrics.mean_squared_error(y_test, pred_y,squared=False)
    print("{} error for k = {}".format(mse,i))
```

```
7.580880154071545 error for k = 11
7.620709624858009 error for k = 12
7.702433441773159 error for k = 13
7.745706188130712 error for k = 14
7.855546909761407 error for k = 15
7.970845764140948 error for k = 16
8.00708692880329 error for k = 17
8.05951400020052 error for k = 18
8.105972848197592 error for k = 19
8.171623447622684 error for k = 20
8.208766061680672 error for k = 21
8.266010100575647 error for k = 22
8.280897264278922 error for k = 23
8.326448746059764 error for k = 24
8.38105978099617 error for k = 25
8.410954693047014 error for k = 26
8.478704509976565 error for k = 27
8.50999986845734 error for k = 28
8.538275555508479 error for k = 29
8.57421797961705 error for k = 30
8.599468444172452 error for k = 31
8.616512206922689 error for k = 32
```

Model Evaluation

```
#Error for KNN Algorithm
mse_Nn= sklearn.metrics.mean_squared_error(y_test, pred_Nn,squared=False)
print("error for K-NN = {}".format(mse_Nn))
```

```
error for K-NN = 7.014927171138291
```

A hyperparameter **controls the learning process** and therefore their values directly impact other parameters of the model such as weights and biases which consequently impacts how well our model performs. The accuracy of any machine learning model is most often improved by fine-tuning these hyperparameters. 15-May-2022

https://www.analyticsvidhya.com › i...

Impact of Hyperparameters on a Deep Learning Model - Analytics Vidhya

Support Vector Machines (SVMs)

SVMs are an important aspect of Machine Learning

- Considered as a black box technique due to presence of unknown parameters.
- Data separation for non-linear classes are done using kernel tricks.
- Parameters are tuned using kernel functions.

Support Vector Machines (SVMs)

SVMs are an important aspect of Machine Learning

- Very flexible working with a variety of data (unstructured, structured and semi-structured)
- Overfitting is very less compared to other models.
- But training time is more if the dataset is large.
- Very popular in healthcare and banking sectors.

Importing all the libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Importing the dataset

```
[ ] dataset = pd.read_csv('Social_Network_Ads.csv')
    dataset.head()
```

Data split for training and testing (75/25)

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

Filtering out columns to retain age and salary columns

```
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values
```

Scaling using Standard Scaler for Normal Distribution

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Building the model using RBF Kernel

```
from sklearn.svm import SVC
classifier_rbf= SVC (kernel = 'rbf', random_state = 0)
classifier_rbf.fit(X_train, y_train)
y_pred_rbf= classifier_rbf.predict(X_test)
```

Printing the confusion matrix

```
from sklearn.metrics import confusion_matrix
cm_rbf=confusion_matrix(y_test,y_pred_rbf)
print(cm_rbf)
```

```
[[64  4]
 [ 3 29]]
```

Classification Report

```
from sklearn.metrics import classification_report
class_report_rbf= classification_report (y_test,y_pred_rbf)
print(class_report_rbf)
```

```
from sklearn.metrics import classification_report
class_report_rbf= classification_report (y_test,y_pred_rbf)
print(class_report_rbf)
```

```
              precision    recall  f1-score   support

           0       0.96      0.94      0.95        68
           1       0.88      0.91      0.89        32

    accuracy                           0.93       100
   macro avg       0.92      0.92      0.92       100
weighted avg       0.93      0.93      0.93       100
```

Random Forest Algorithm

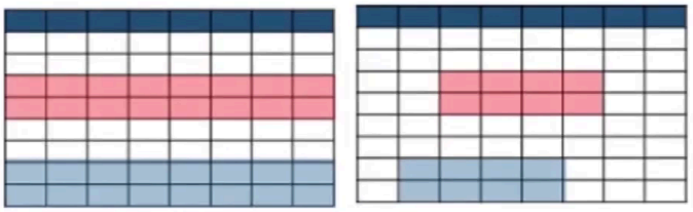
A forest full of trees!

- The algorithm contains a bundle of decision trees. Hence, the name!
- Decision trees have high variance and low bias (unstable output)
- Takes in data, creates branches, and presents each output as a leaf.

Random Forest Algorithm

A forest full of trees!

Consider two samples
(represented by 2 colors)



Bagging vs Random Forest

Random Forest Algorithm

A forest full of trees!

- Works both for classification and regression problems
- Automation of missing values is possible
- Required computation power for complex trees

Importing libraries and dataset

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
df = pd.read_csv('Salaries.csv')
df.head()
```

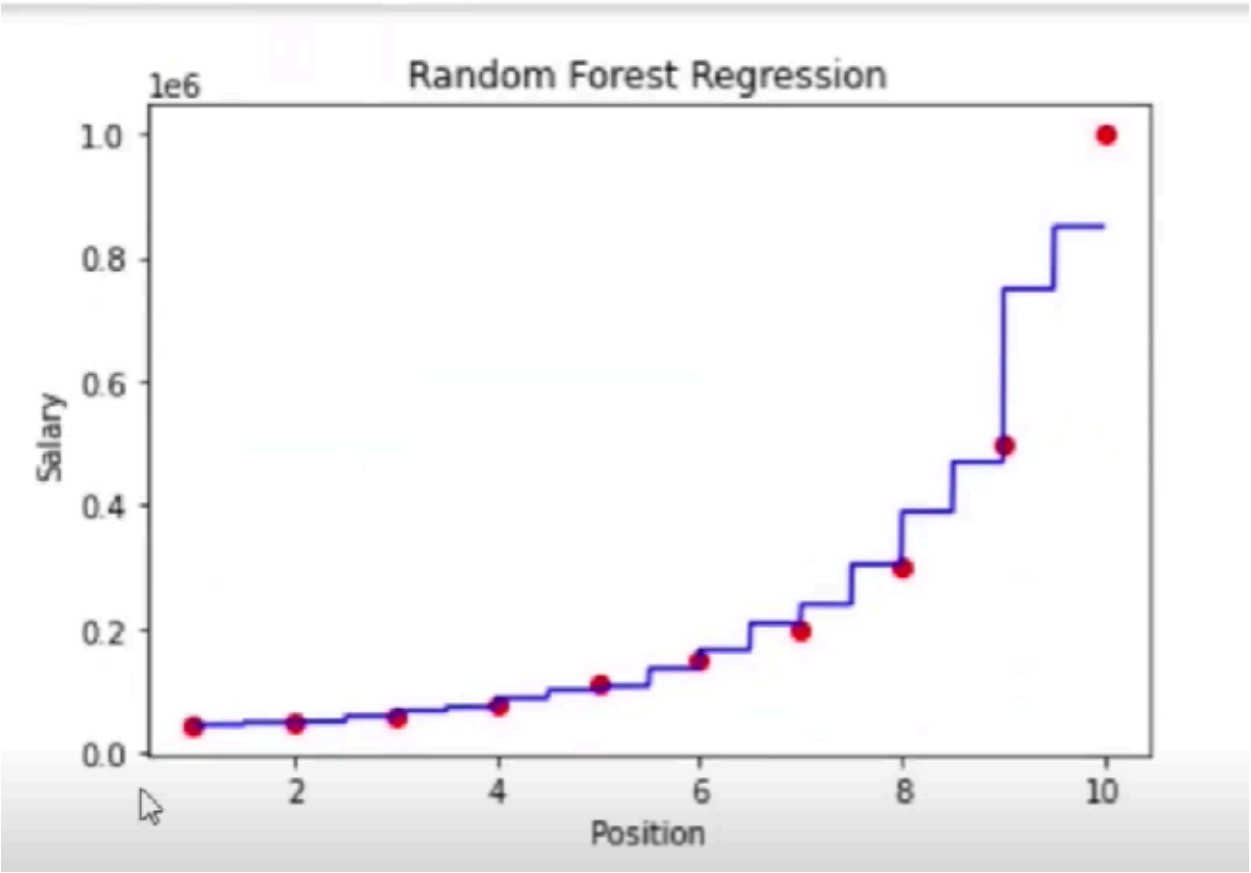
	Position	Level	Salary
0	Business Analyst	1	45000
1	Junior Consultant	2	50000
2	Senior Consultant	3	60000
3	Manager	4	80000
4	Country Manager	5	110000
5	Region Manager	6	150000
6	Partner	7	200000
7	Senior Partner	8	300000

0	Business Analyst	1	45000
1	Junior Consultant	2	50000
2	Senior Consultant	3	60000
3	Manager	4	80000
4	Country Manager	5	110000
5	Region Manager	6	150000
6	Partner	7	200000
7	Senior Partner	8	300000
8	C-level	9	500000
9	CEO	10	1000000

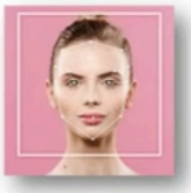
```
X =df.iloc[:, 1:2].values
y =df.iloc[:, 2].values
```

Model fitting with 10 trees

```
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(n_estimators = 10, random_state = 0)
model.fit(X, y)
```


Random Forest Regression
1e6
1.0
0.8
0.6
0.4
0.2
0.0
Salary
2
4
6
8
10
Position

Real World Applications of Machine Learning



Face Recognition

Siri and Cortana

Healthcare Industry

Weather Forecasting

Produce a Web Series

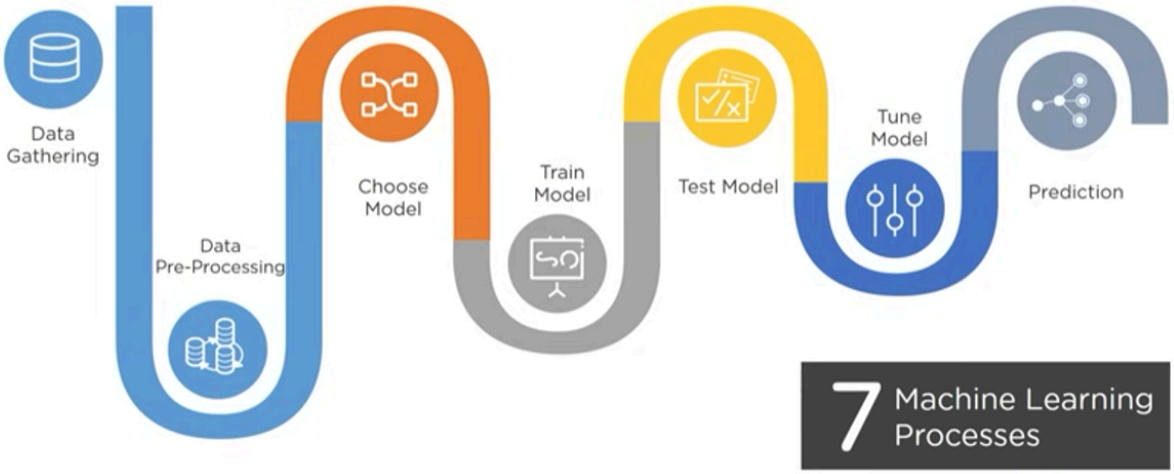
What is Machine Learning?

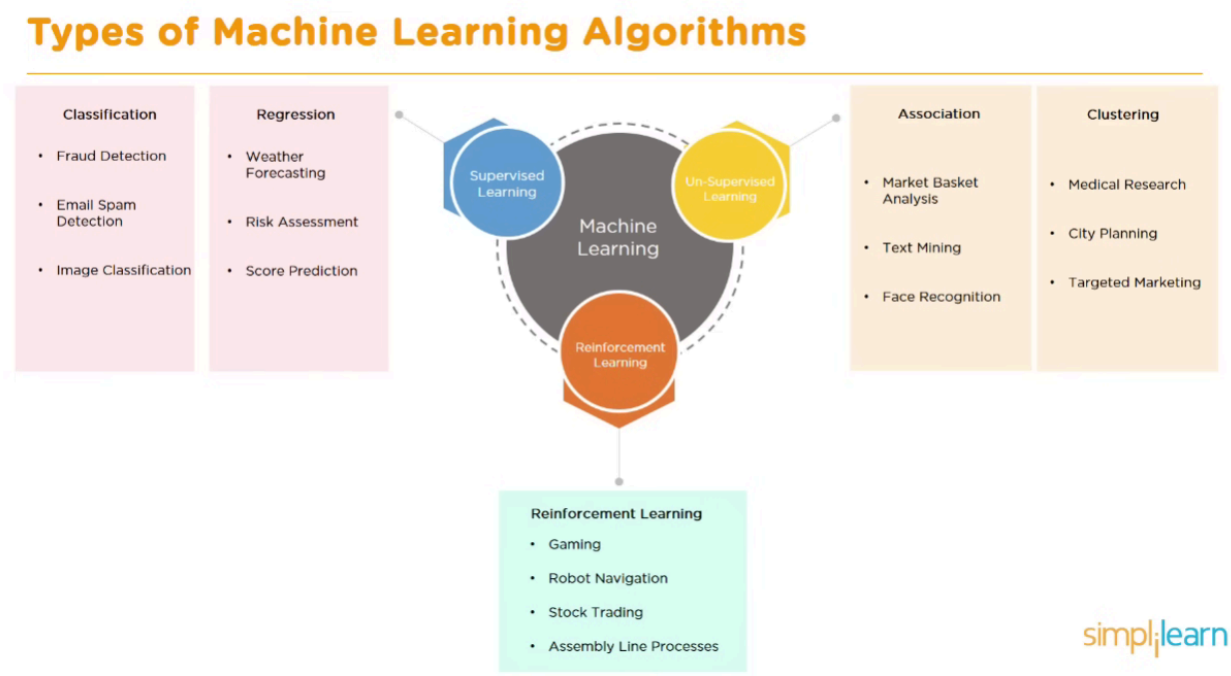
Machine Learning is the science of making computers learn and act like humans by feeding data and information without being explicitly programmed.

What is Machine Learning?

Machine Learning is the science of making computers learn and act like humans by feeding data and information without being explicitly programmed.

Processes involved in Machine Learning





Supervised Learning - Labeled data

UnSupervised Learning - Non Labeled data

Reinforcement Learning - Training with Regards

If you have labeled information Go with the supervised machine learning

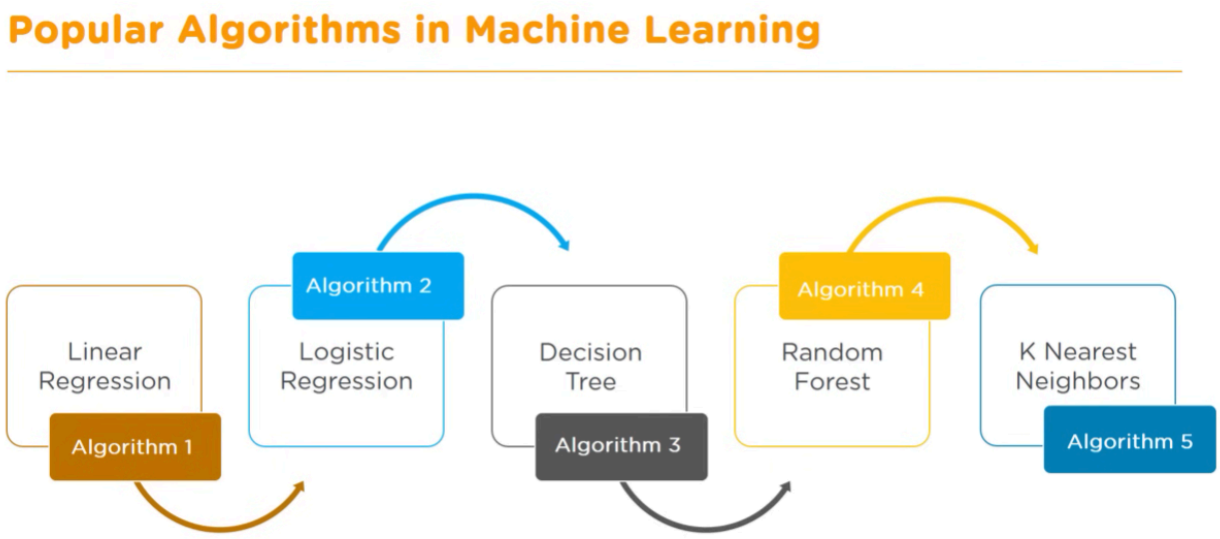
Regression - target would be the value

Regression is a supervised machine learning technique which is used to predict continuous values. The ultimate goal of the regression algorithm is to plot a best-fit line or a curve between the data. The three main metrics that are used for evaluating the trained regression model are variance, bias and error.

Classification - target is to predict the class

Classification is a supervised machine learning method where the model tries to predict the correct label of a given input data. In classification, the model is fully trained using

the training data, and then it is evaluated on test data before being used to perform prediction on new unseen data.



Regression Analysis in Machine learning

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

Regression analysis helps us to understand how the value of the dependent variable is changing corresponding to an independent variable when other independent variables are held fixed. It predicts continuous/real values such as **temperature, age, salary, price,** etc.

https://www.javatpoint.com/regression-analysis-in-machine-learning

Regression is a supervised learning technique which helps in finding the correlation between variables and enables us to predict the continuous output variable based on the one or more predictor variables