The Complete Data Science Study Roadmap

18 July 2023 17:54

1. **Mathematics**

   Part 1:

   i. Linear Algebra
   ii. Analytic Geometry
   iii. Matrix
   iv. Vector Calculus
   v. Optimization

   Part 2:

   i. Regression
   ii. Dimensionality Reduction
   iii. Density Estimation
   iv. Classification

3. **Statistics**
   1. Introduction to Statistics
   2. Data Description
   3. Random Samples
   4. Sampling Distribution
   5. Parameter Estimation
   6. Hypotheses Testing (Python | R)
   7. ANOVA (Python | R)
   8. Reliability Engineering
   9. Stochastic Process
   10. Computer Simulation
   11. Design of Experiments
   12. Simple Linear Regression

5. **Machine Learning**

   Introduction:

   i. How Model Works
   ii. Basic Data Exploration
   iii. First ML Model
   iv. Model Validation
   v. Underfitting & Overfitting
   vi. Random Forests (Python | R)
   vii. scikit-learn

   Intermediate:

   i. Handling Missing Values
   ii. Handling Categorical Variables
   iii. Pipelines
   iv. Cross-Validation (R)
   v. XGBoost (Python | R)
   vi. Data Leakage

2. **Probability**
   1. Introduction to Probability
   2. 1D Random Variable
   3. The function of One Random Variable
   4. Joint Probability Distribution
   5. Discrete Distribution
   6. Binomial (Python | R)
   7. Bernoulli
   8. Geometric etc
   9. Continuous Distribution
   10. Uniform
   11. Exponential
   12. Gamma
   13. Normal Distribution (Python | R)

4. **Programming**

   Python:
   1. Python Basics
   2. List
   3. Set
   4. Tuples
   5. Dictionary
   6. Function, etc.
   7. NumPy
   8. Pandas
   9. Matplotlib/Seaborn, etc.

   R:
   1. R Basics
   2. Vector
   3. List
   4. Data Frame
   5. Matrix
   6. Array
   7. Function, etc.
   8. dplyr
   9. ggplot2
   10. Tidyr
   11. Shiny, etc.

   Database:
   1. SQL
   2. MongoDB

   Other:
   1. Data Structure
   2. Time Complexity
   3. Web Scraping (Python | R)
   4. Linux
   5. Git

6. Deep Learning

i. Artificial Neural Network
ii. Convolutional Neural Network
iii. Recurrent Neural Network
iv. TensorFlow
v. Keras
vi. PyTorch
vii. A Single Neuron
viii. Deep Neural Network
ix. Stochastic Gradient Descent
x. Overfitting and Underfitting
xi. Dropout Batch Normalization
xii. Binary Classification

7. Feature Engineering

1. Baseline Model
2. Categorical Encodings
3. Feature Generation
4. Feature Selection

8. Natural Language Processing

a. Text Classification
b. Word Vectors

9. Data Visualization Tools

i. Excel VBA
ii. BI (Business Intelligence):
iii. Tableau
iv. Power BI
v. Qlik View
vi. Qlik Sense

9.Deployment

i. Microsoft Azure
ii. Heroku
iii. Google Cloud Platform
iv. Flask
v. Django

ML Algorithm - Dataset

08 July 2023 12:21

What is Machine Learning?
In basic terms, ML is the process of training a piece of software, called a model, to make useful predictions or generate content from data.

For example, suppose we wanted to create an app to predict rainfall. We could use either a traditional approach or an ML approach. Using a traditional approach, we'd create a physics-based representation of the Earth's atmosphere and surface, computing massive amounts of fluid dynamics equations. This is incredibly difficult.

Using an ML approach, we would give an ML model enormous amounts of weather data until the ML model eventually learned the mathematical relationship between weather patterns that produce differing amounts of rain. We would then give the model the current weather data, and it would predict the amount of rain.

To Store Dataset:

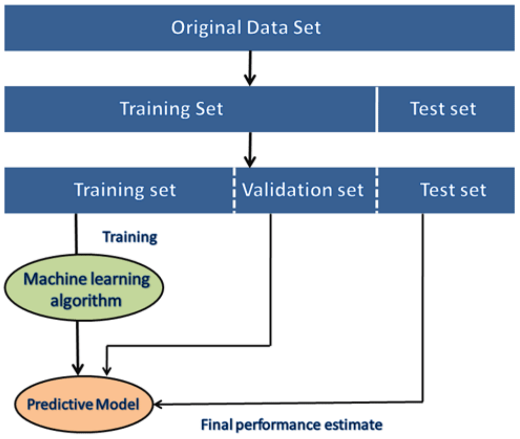
1. Tabular: CSV format
2. Tree like structure data: JSON file

Types of data in datasets

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

Need of Dataset

- Training dataset
- Test Dataset



Popular Sources of getting datasets:

1. Kaggle
2. UCI Machine Learning Repository
3. Google dataset Search Engine
4. Government dataset
5. AWS

Data Pre-processing

08 July 2023 13:49

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

Why do we need Data Pre-processing?

A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models.

It involves below steps:

1. *Getting the dataset*
2. *Importing libraries*
3. *Importing datasets*
4. *Finding Missing Data*
5. *Encoding Categorical Data*
6. *Splitting dataset into training and test set*
7. *Feature scaling*

1. Get dataset from various sources

- Kaggle
- UCI Machine Learning Repository
- Google dataset Search Engine
- Government dataset
- AWS

2. Importing Libraries

- NumPy
- Pandas
- Matplotlib
- Seaborn
- Scikit Learn

3. Importing dataset

- Read_csv() - function to read csv dataset

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

- Extracting dependent and independent variables:
  iloc[ ] method of Pandas library : It is used to extract the required rows and columns from the dataset.

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

4. Handling Missing Values
   I. By deleting particular row
   II.By calculating the mean
   Note: use scikit learn library

```
#handling missing data (Replacing missing data with the mean value)
from sklearn.preprocessing import Imputer
```

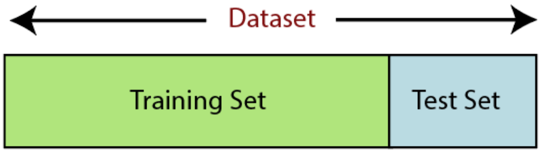
```
imputer= Imputer(missing_values ='NaN', strategy='mean', axis = 0)
#Fitting imputer object to the independent variables x.
imputerimputer= imputer.fit(x[:, 1:3])
#Replacing missing data with the calculated mean value
x[:, 1:3]= imputer.transform(x[:, 1:3])
```

5. Encoding Categorical Data

Having categories in the dataset

```
Use: LabelEncoder() class from preprocessing library

#Catgorical data
#for Country Variable
from sklearn.preprocessing import LabelEncoder
label_encoder_x= LabelEncoder()
x[:, 0]= label_encoder_x.fit_transform(x[:, 0])
```

6. Split datasets into Train and Test



Training Set: A subset of dataset to train the machine learning model, and we already know the output.
Test set: A subset of dataset to test the machine learning model, and by using the test set, model predicts the output.

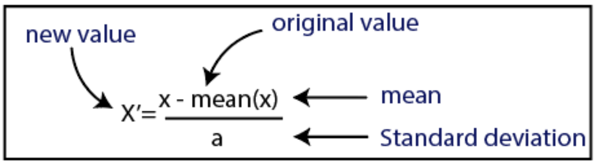
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.
2, random_state=0)
```

7. Feature Scaling

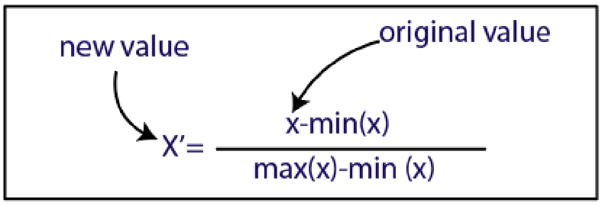
Feature scaling is the final step of data pre-processing in machine learning. It is a technique to standardize the independent variables of the dataset in a specific range. In feature scaling, we put our variables in the same range and in the same scale so that no any variable dominate the other variable.

There are 2 ways to do feature scaling:

1.Standardization



2.Normalization



import ***StandardScaler*** class of ***sklearn.preprocessing*** library as:
from sklearn.preprocessing import StandardScaler

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

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

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

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

Code Example:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('Dataset.csv')

#Extracting Independent Variable
x= data_set.iloc[:, :-1].values

#Extracting Dependent variable
y= data_set.iloc[:, 3].values

#handling missing data(Replacing missing data with the mean value)
from sklearn.preprocessing import Imputer
imputer= Imputer(missing_values ='NaN', strategy='mean', axis = 0)

#Fitting imputer object to the independent varibles x.
imputerimputer= imputer.fit(x[:, 1:3])

#Replacing missing data with the calculated mean value
x[:, 1:3]= imputer.transform(x[:, 1:3])

#for Country Variable
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
label_encoder_x= LabelEncoder()
x[:, 0]= label_encoder_x.fit_transform(x[:, 0])

#Encoding for dummy variables
onehot_encoder= OneHotEncoder(categorical_features= [0])
x= onehot_encoder.fit_transform(x).toarray()

#encoding for purchased variable
labelencoder_y= LabelEncoder()
y= labelencoder_y.fit_transform(y)

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

#Feature Scaling of datasets
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

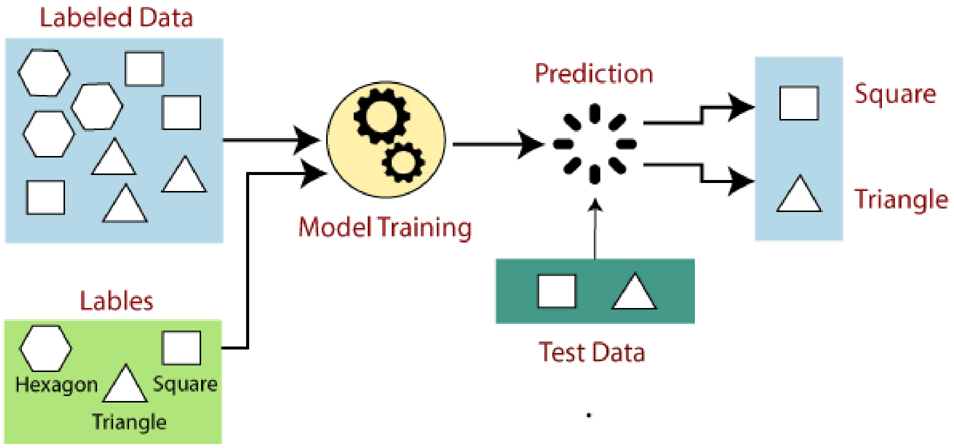
In the above code, we have included all the data pre-processing steps together. But there are some steps or lines of code which are not necessary for all machine learning models. So we can exclude them from our code to make it reusable for all models.

Supervised ML

08 July 2023 14:57

Supervised learning is the types of machine learning in which machines are trained using well "labelled" training data, and on basis of that data, machines predict the output. The labelled data means some input data is already tagged with the correct output.
The aim of a supervised learning algorithm is to **find a mapping function to map the input variable(x) with the output variable(y)**.

How Supervised Learning Works?

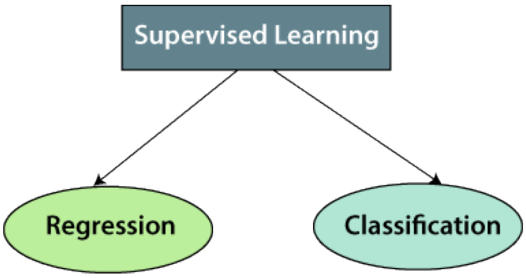


Steps Involved in Supervised Learning:

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

Types of supervised Machine learning Algorithms:

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



1. Regression

Regression algorithms are used if there is a relationship between the input variable and the output variable. It is used for the prediction of continuous variables, such as Weather forecasting, Market Trends, etc.

Some popular Regression algorithms which come under supervised learning:

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

2. Classification

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

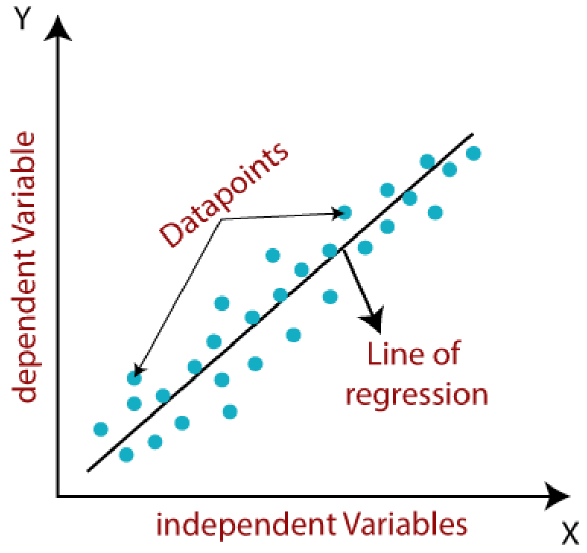
Some popular classification algorithms which come under supervised learning:

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

Linear Regression

08 July 2023 17:05

1. Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression.
2. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.
3. The linear regression model provides a sloped straight line representing the relationship between the variables.



Mathematically, we can represent a linear regression as:

$y= a_0+a_1x+ \varepsilon$

Here,

Y= Dependent Variable (Target Variable)
X= Independent Variable (predictor Variable)
a0= intercept of the line (Gives an additional degree of freedom)
a1 = Linear regression coefficient (scale factor to each input value).
ε = random error
The values for x and y variables are training datasets for Linear Regression model representation.

Types of Linear Regression

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

- **Simple Linear Regression:**

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

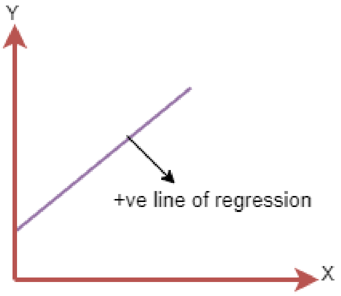
- **Multiple Linear regression:**

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

Linear Regression Line

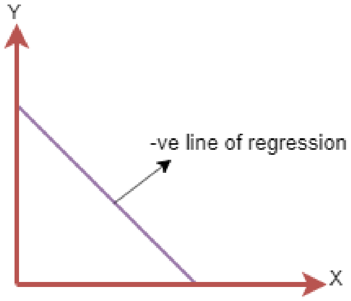
A linear line showing the relationship between the dependent and independent variables is called a **regression line**. A regression line can show two types of relationship:

- **Positive Linear Relationship:**
  If the dependent variable increases on the Y-axis and independent variable increases on X-axis, then such a relationship is termed as a Positive linear relationship.



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

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



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

Finding the best fit line:

When working with linear regression, our main goal is to find the best fit line that means the error between predicted values and actual values should be minimized. The best fit line will have the least error.
The different values for weights or the coefficient of lines (a_0, a_1) gives a different line of regression, so we need to calculate the best values for a_0 and a_1 to find the best fit line, so to calculate this we use cost function.

Cost function-

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

For Linear Regression, we use the **Mean Squared Error (MSE)** cost function, which is the average of

squared error occurred between the predicted values and actual values. It can be written as:
For the above linear equation, MSE can be calculated as:

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

Where,
N=Total number of observation
Yi = Actual value
$(a1x_i+a_0)$= Predicted value.

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

Gradient Descent:

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

Model Performance:

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

- R-squared is a statistical method that determines the goodness of fit.
- It measures the strength of the relationship between the dependent and independent variables on a scale of 0-100%.
- The high value of R-square determines the less difference between the predicted values and actual values and hence represents a good model.
- It is also called a **coefficient of determination,** or **coefficient of multiple determination** for multiple regression.
- It can be calculated from the below formula:

$$\text{R-squared}=\frac{\text{Explained variation}}{\text{Total Variation}}$$

Assumptions of Linear Regression

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

- **Linear relationship between the features and target:**
  Linear regression assumes the linear relationship between the dependent and independent variables.
- **Small or no multicollinearity between the features:**

Multicollinearity means high-correlation between the independent variables. Due to multicollinearity, it may difficult to find the true relationship between the predictors and target variables. Or we can say, it is difficult to determine which predictor variable is affecting the target variable and which is not. So, the model assumes either little or no multicollinearity between the features or independent variables.

- **Homoscedasticity Assumption:**
  Homoscedasticity is a situation when the error term is the same for all the values of independent variables. With homoscedasticity, there should be no clear pattern distribution of data in the scatter plot.
- **Normal distribution of error terms:**
  Linear regression assumes that the error term should follow the normal distribution pattern. If error terms are not normally distributed, then confidence intervals will become either too wide or too narrow, which may cause difficulties in finding coefficients.
  It can be checked using the **q-q plot**. If the plot shows a straight line without any deviation, which means the error is normally distributed.
- **No autocorrelations:**
  The linear regression model assumes no autocorrelation in error terms. If there will be any correlation in the error term, then it will drastically reduce the accuracy of the model. Autocorrelation usually occurs if there is a dependency between residual error

1.Simple Linear regression

09 July 2023 15:20

- Simple Linear Regression is a type of Regression algorithms that models the relationship between a dependent variable and a single independent variable.
- The relationship shown by a Simple Linear Regression model is linear or a sloped straight line, hence it is called Simple Linear Regression.
- Simple Linear Regression is that the ***dependent variable must be a continuous/real value***. However, the independent variable can be measured on continuous or categorical values.

Simple Linear regression algorithm has mainly two objectives:

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

Simple Linear Regression Model:

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

$y= a_0+a_1x+ \varepsilon$

Where,

a0= *It is the intercept of the Regression line (can be obtained putting x=0)*

a1= *It is the slope of the regression line, which tells whether the line is increasing or decreasing.*

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

Implementation of Simple Linear Regression Algorithm

Step-1: Data Pre-processing

```
#import the required libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#Load the dataset
data_set= pd.read_csv('Salary_Data.csv')

#extract the dependent and independent variables from the given dataset
x= data_set.iloc[:, :-1].values
y= data_set.iloc[:, 1].values

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

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

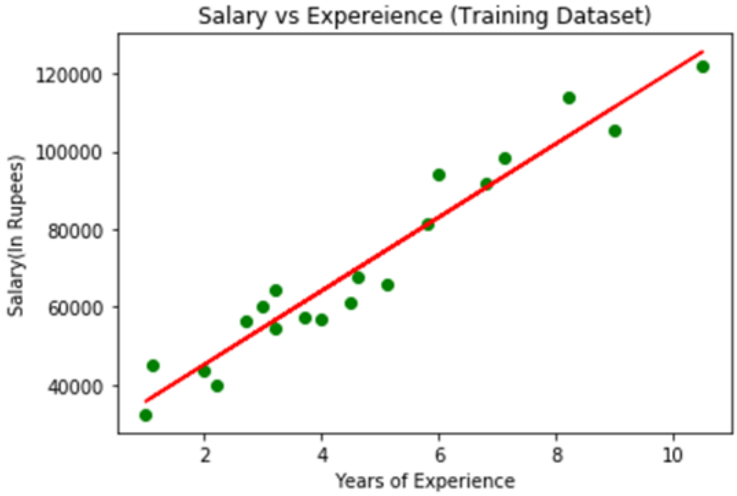
```
#Fitting the Simple Linear Regression model to the training dataset
from sklearn.linear_model import LinearRegression
regressor= LinearRegression()
regressor.fit(x_train, y_train)
```

Step: 3. Prediction of test set result:

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

Step: 4. visualizing the Training set results:

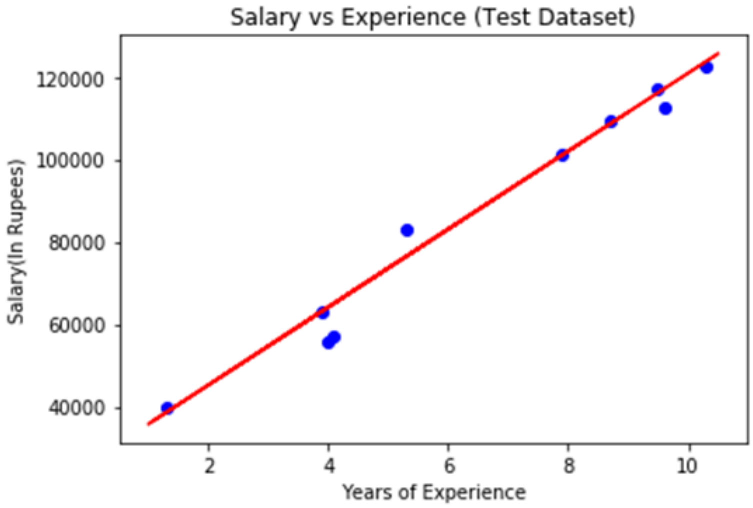
```
mtp.scatter(x_train, y_train, color="green")
mtp.plot(x_train, x_pred, color="red")
mtp.title("Salary vs Experience (Training Dataset)")
mtp.xlabel("Years of Experience")
mtp.ylabel("Salary(In Rupees)")
mtp.show()
```



- *we can see in the above plot, most of the observations are close to the regression line, hence our model is good for the training set.*

Step: 5. visualizing the Test set results:

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



a) *There are observations given by the blue colour, and prediction is given* *by the red regression line.*

b) *Most of the observations are close to the regression line.*

c) *Our Simple Linear Regression is a good model and able to make good predictions.*

2.Multilinear Linear regression

09 July 2023 15:48

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

Some key points about MLR:

- For MLR, the dependent or target variable(Y) must be the continuous/real, but the predictor or independent variable may be of continuous or categorical form.
- Each feature variable must model the linear relationship with the dependent variable.
- MLR tries to fit a regression line through a multidimensional space of data-points.

1. Y= b<sub>0</sub>+b<sub>1</sub>x<sub>1</sub>+ b<sub>2</sub>x<sub>2</sub>+ b<sub>3</sub> x<sub>3</sub>+...... bnxn .... (a)
   Where,
   Y= Output/Response variable
   b_0, b_1, b_2, b_3 , b_n....= Coefficients of the model.
   x_1, x_2, x_3, x_4,...= Various Independent/feature variable

Assumptions for Multiple Linear Regression:

- A **linear relationship** should exist between the Target and predictor variables.
- The regression residuals must be **normally distributed**.
- MLR assumes little or **no multicollinearity** (correlation between the independent variable) in data.

Implementation of Multiple Linear Regression model using Python:

Problem Description:
We have a dataset of **50 start-up companies**. This dataset contains five main information: **R&D Spend, Administration Spend, Marketing Spend, State, and Profit for a financial year**. Our goal is to create a model that can easily determine which company has a maximum profit, and which is the most affecting factor for the profit of a company.
Since we need to find the Profit, so it is the dependent variable, and the other four variables are independent variables. Below are the main steps of deploying the MLR model:

1. **Data Pre-processing Steps**
2. **Fitting the MLR model to the training set**
3. **Predicting the result of the test set**

Step-1: Data Pre-processing Step:

Importing libraries:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

Importing dataset:

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

Extracting dependent and independent Variables:

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

Encoding Dummy Variables:

```
#Catgorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_x= LabelEncoder()
x[:, 3]= labelencoder_x.fit_transform(x[:,3])
onehotencoder= OneHotEncoder(categorical_features= [3])
x= onehotencoder.fit_transform(x).toarray()
```

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

```
#avoiding the dummy variable trap:
x = x[:, 1:]
```

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

```
# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.2, random_state=0)
```

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

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

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

```
#Fitting the MLR model to the training set:
from sklearn.linear_model import LinearRegression
regressor= LinearRegression()
regressor.fit(x_train, y_train)
```

Step: 3- Prediction of Test set results:

```
#Predicting the Test set result;
y_pred= regressor.predict(x_test)
```

- ➢ *By test our model by comparing the predicted values and test set values.*

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

Output: The score is:

```
Train Score:  0.9501847627493607
Test Score:  0.9347068473282446
```

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

3.Polynomial Regression

09 July 2023 16:56

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

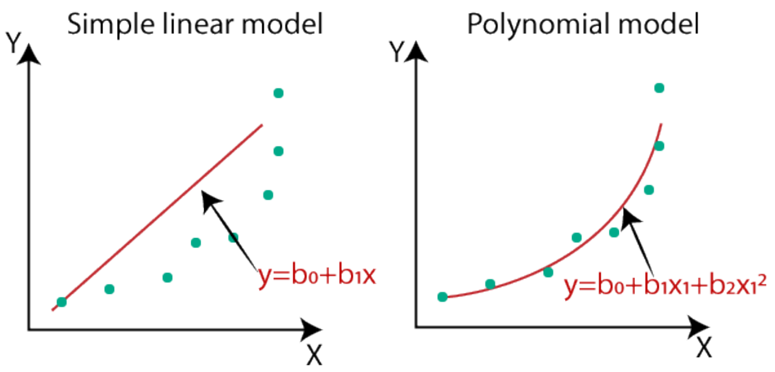
y= b0+b1x1+ b2x12+ b2x13+...... Bnx1n

In Polynomial regression, the original features are converted into Polynomial features of required degree (2,3,..,n) and then modelled using a linear model.

Need for Polynomial Regression:

The need of Polynomial Regression in ML can be understood in the below points:

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



if the datasets are arranged in a non-linear fashion, then we should use the Polynomial Regression model instead of Simple Linear Regression.

Equation of the Polynomial Regression Model:

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

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

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

Implementation of Polynomial Regression using Python:

Problem Description: There is a Human Resource company, which is going to hire a new candidate. The candidate has told his previous salary 160K per annum, and the HR have to check whether he is telling the truth or bluff. So to identify this, they only have a dataset of his previous company in which the salaries of the top 10 positions are mentioned with their levels. By checking the dataset available, we have found that there is a **non-linear relationship between the Position levels and the salaries**. Our goal is to build a **Bluffing detector regression** model, so HR can hire an honest candidate.

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

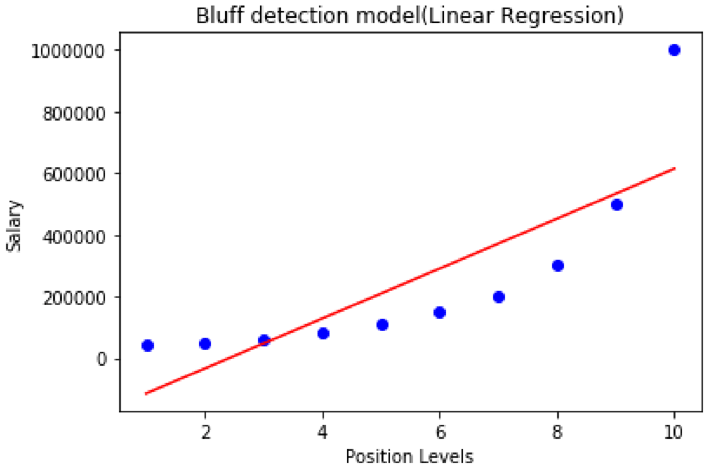
1.

```
#importing datasets
data_set= pd.read_csv('Position_Salaries.csv')
```

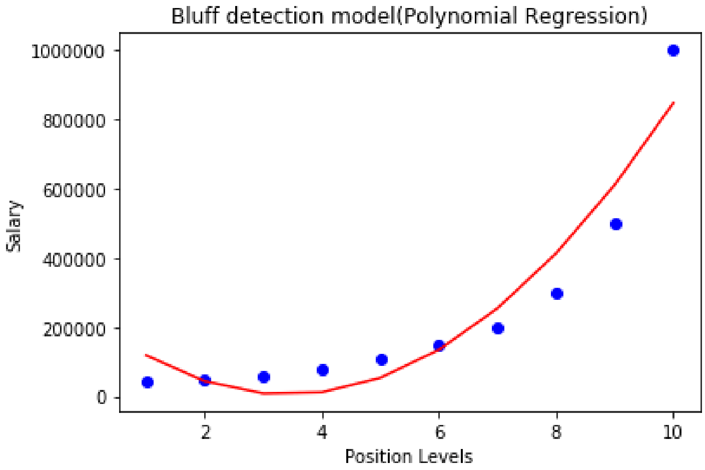
2.

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

#Fitting the Linear Regression to the dataset
from sklearn.linear_model import LinearRegression
lin_regs= LinearRegression()
lin_regs.fit(x,y)
```

```
#Fitting the Polynomial regression to the dataset
from sklearn.preprocessing import PolynomialFeatures
poly_regs= PolynomialFeatures(degree= 2)
x_poly= poly_regs.fit_transform(x)
lin_reg_2 =LinearRegression()
lin_reg_2.fit(x_poly, y)


#Visulaizing the result for Linear Regression model
mtp.scatter(x,y,color="blue")
mtp.plot(x,lin_regs.predict(x), color="red")
mtp.title("Bluff detection model(Linear Regression)")
mtp.xlabel("Position Levels")
mtp.ylabel("Salary")
mtp.show()
```



```
#Visulaizing the result for Polynomial Regression
mtp.scatter(x,y,color="blue")
mtp.plot(x, lin_reg_
2.predict(poly_regs.fit_transform(x)), color="red")
mtp.title("Bluff detection model(Polynomial Regression)")
mtp.xlabel("Position Levels")
mtp.ylabel("Salary")
mtp.show()
```



```
#predict linear regression result
lin_pred = lin_regs.predict([[6.5]])
print(lin_pred)
```

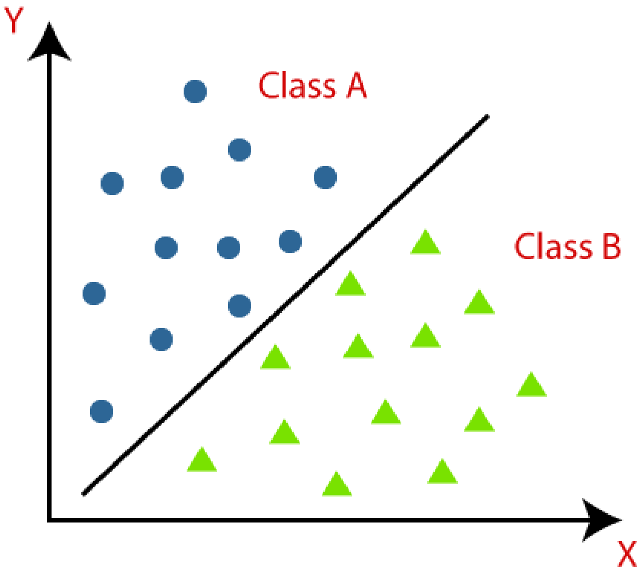
```
#predict polynomial regression
poly_pred = lin_reg_2.predict(poly_regs.fit_transform([[6.5]]))
print(poly_pred)
```

Classification Algorithm

09 July 2023 17:17

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

```
y=f(x), where y = categorical output
```



There are two types of Classifications:

- **Binary Classifier:** If the classification problem has only two possible outcomes, then it is called as Binary Classifier.

  Examples: YES or NO, MALE or FEMALE, SPAM or NOT SPAM, CAT or DOG, etc.
- **Multi-class Classifier:** If a classification problem has more than two outcomes, then it is called as Multi-class Classifier.

  Example: Classifications of types of crops, Classification of types of music.

Learners in Classification Problems:

In the classification problems, there are two types of learners:

1. **Lazy Learners:** Lazy Learner firstly stores the training dataset and wait until it receives the test dataset. In Lazy learner case, classification is done on the basis of the most related data stored in the training dataset. It takes less time in training but more time for predictions.

   Example: K-NN algorithm, Case-based reasoning
2. **Eager Learners:** Eager Learners develop a classification model based on a training dataset before receiving a test dataset. Opposite to Lazy learners, Eager Learner takes more time in learning, and less time in prediction.

   Example: Decision Trees, Naïve Bayes, ANN.

Types of ML Classification Algorithms:

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

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

Evaluating a Classification model:

Once our model is completed, it is necessary to evaluate its performance; either it is a Classification or Regression model. So for evaluating a Classification model, we have the following ways:

1. Log Loss or Cross-Entropy Loss:

- It is used for evaluating the performance of a classifier, whose output is a probability value between the 0 and 1.
- For a good binary Classification model, the value of log loss should be near to 0.
- The value of log loss increases if the predicted value deviates from the actual value.
- The lower log loss represents the higher accuracy of the model.
- For Binary classification, cross-entropy can be calculated as:

1. ?(ylog(p)+(1?y)log(1?p))
   Where,
   y= Actual output, p= predicted output.

2. Confusion Matrix:

- The confusion matrix provides us a matrix/table as output and describes the performance of the model.
- It is also known as the error matrix.
- The matrix consists of predictions result in a summarized form, which has a total number of correct predictions and incorrect predictions.
- The matrix looks like as below table:

	Actual Positive	**Actual Negative**
Predicted Positive	True Positive	False Positive
Predicted Negative	False Negative	True Negative

$$\text{Accuracy} = \frac{\text{TP+TN}}{\text{Total Population}}$$

3. AUC-ROC curve:

- ROC curve stands for **Receiver Operating Characteristics Curve** and AUC stands for **Area Under the Curve**.
- It is a graph that shows the performance of the classification model at different thresholds.
- To visualize the performance of the multi-class classification model, we use the AUC-ROC Curve.
- The ROC curve is plotted with TPR and FPR, where TPR (True Positive Rate) on Y-axis and FPR(False Positive Rate) on X-axis.

Use cases of Classification Algorithms

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

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

1. Logistic Regression

16 July 2023 17:33

- Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.
- Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas Logistic regression is used for solving the classification problems.
- Logistic regression uses the concept of predictive modelling as regression; therefore, it is called logistic regression, but is used to classify samples; Therefore, it falls under the classification algorithm.

Logistic Function (Sigmoid Function):

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

Assumptions for Logistic Regression:

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

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

- We know the equation of the straight line can be written as:
- In Logistic Regression y can be between 0 and 1 only, so for this let's divide the above equation by (1-y):
- But we need range between -[infinity] to +[infinity], then take logarithm of the equation it will become:

The above equation is the final equation for Logistic Regression.

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

- Binomial: In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.

- Multinomial: In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"
- Ordinal: In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".

Python Implementation of Logistic Regression (Binomial)

Example: There is a dataset given which contains the information of various users obtained from the social networking sites. There is a car making company that has recently launched a new SUV car. So the company wanted to check how many users from the dataset, wants to purchase the car.
For this problem, we will build a Machine Learning model using the Logistic regression algorithm. In this problem, we will predict the purchased variable (Dependent Variable) by using age and salary (Independent variables).

```
#Data Pre-procesing Step
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('user_data.csv')

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

# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25,
random_state=0)

#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)

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

LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=0, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)

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

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

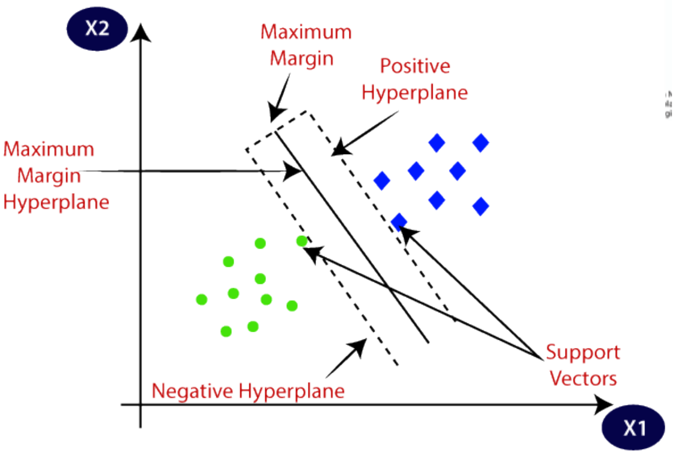
```
#Visualizing the training set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop =
x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1,
step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Logistic Regression (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()

#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop =
x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1,
step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Logistic Regression (Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

2. Support Vector Machine Algorithm

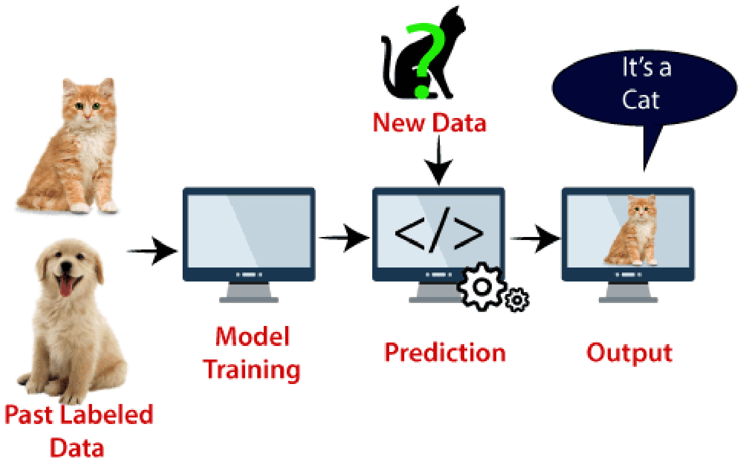
16 July 2023 17:57

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

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



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



Types of SVM

SVM can be of two types:

Linear SVM: Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.

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

Hyperplane and Support Vectors in the SVM algorithm:

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

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

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

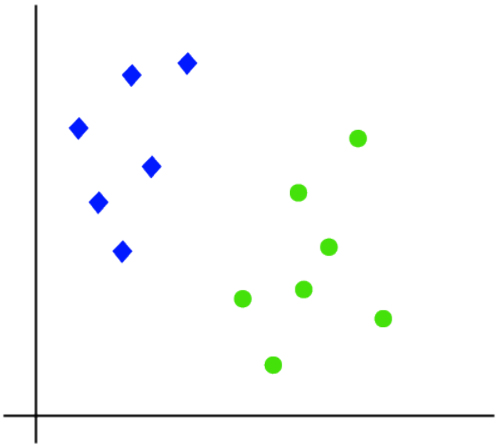
Support Vectors:

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

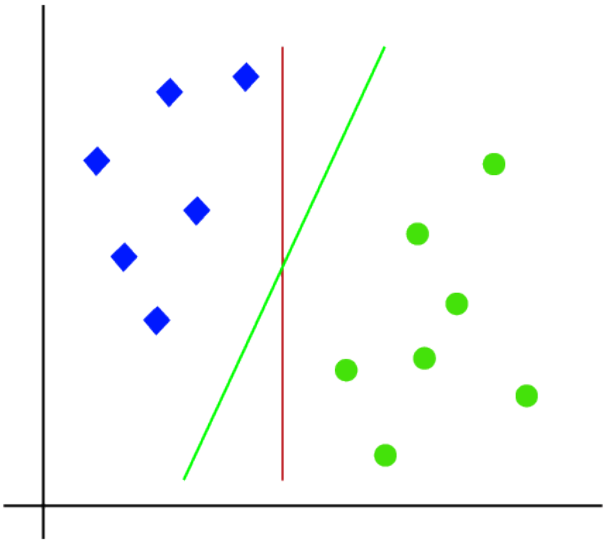
How does SVM works?

Linear SVM:

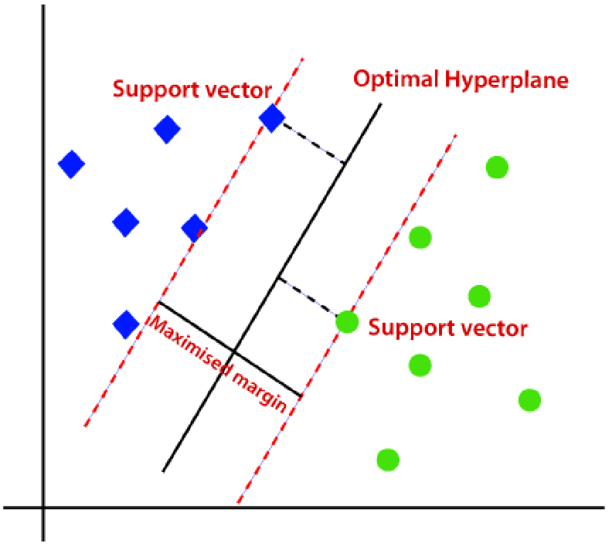
The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image:



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

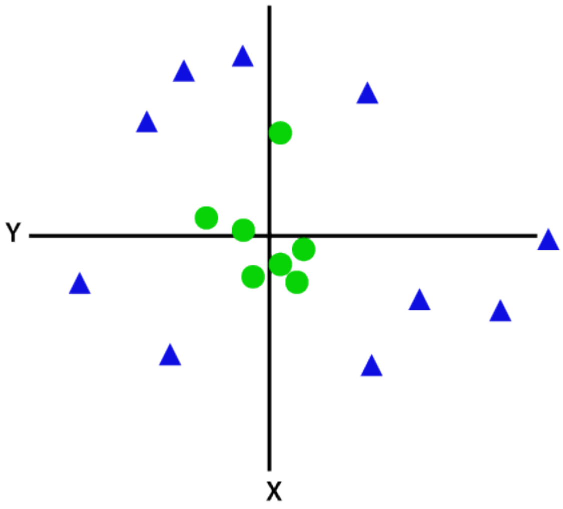


Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a hyperplane. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is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



Non-Linear SVM:

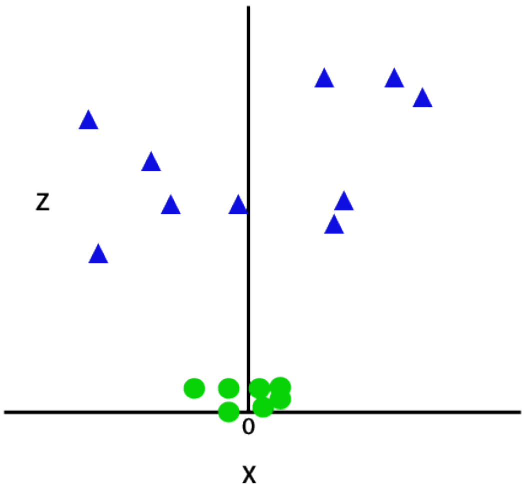
If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:



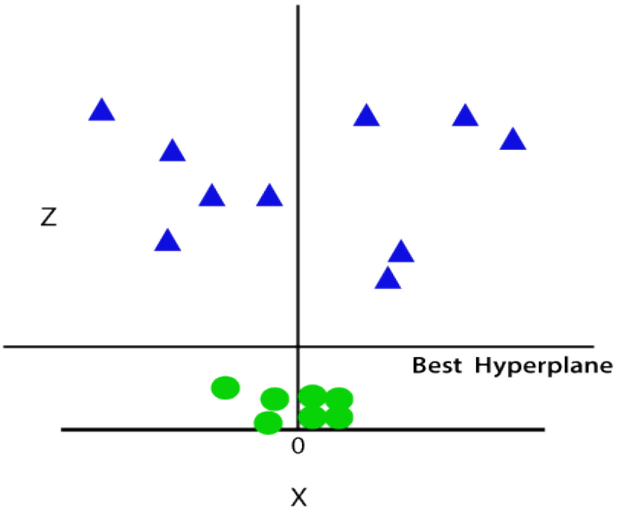
So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

z=x2 +y2

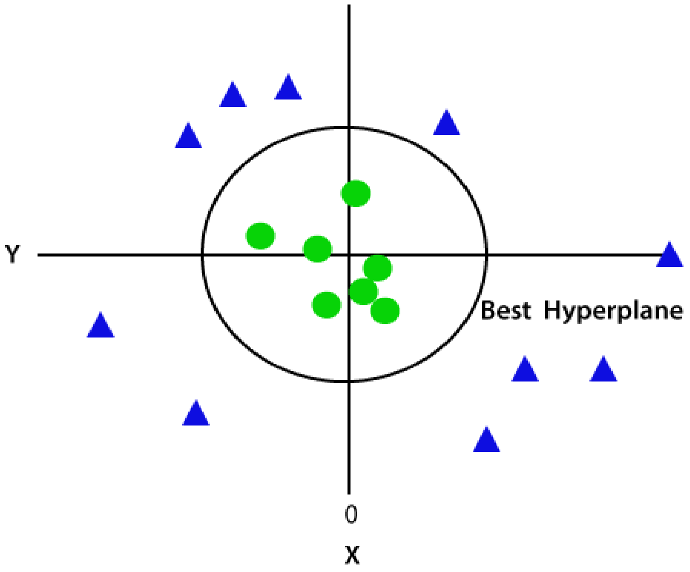
By adding the third dimension, the sample space will become as below image:



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



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



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

Python Implementation of Support Vector Machine

```
#Data Pre-processing Step
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('user_data.csv')

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

# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25,
random_state=0)
#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)

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

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

#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
```

```
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop =
x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1,
step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green')))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('SVM classifier (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()

#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop =
x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1,
step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('SVM classifier (Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

3. Naïve Bayes Classifier Algorithm

16 July 2023 18:50

1. Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.
2. It is mainly used in text classification that includes a high-dimensional training dataset.
3. Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
4. It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.
5. Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

Why is it called Naïve Bayes?

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

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

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

The formula for Bayes' theorem is given as:
Where,

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

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

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

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

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

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

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

Advantages of Naïve Bayes Classifier:

1. Naïve Bayes is one of the fast and easy ML algorithms to predict a class of datasets.
2. It can be used for Binary as well as Multi-class Classifications.
3. It performs well in Multi-class predictions as compared to the other Algorithms.
4. It is the most popular choice for text classification problems.

Disadvantages of Naïve Bayes Classifier:

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

Applications of Naïve Bayes Classifier:

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

Types of Naïve Bayes Model:

There are three types of Naive Bayes Model, which are given below:

Gaussian: The Gaussian model assumes that features follow a normal distribution. This means if predictors take continuous values instead of discrete, then the model assumes that these values are sampled from the Gaussian distribution.

Multinomial: The Multinomial Naïve Bayes classifier is used when the data is multinomial distributed. It is primarily used for document classification problems, it means a particular document belongs to which category such as Sports, Politics, education, etc.

The classifier uses the frequency of words for the predictors.
Bernoulli: The Bernoulli classifier works similar to the Multinomial classifier, but the predictor variables are the independent Booleans variables. Such as if a particular word is present or not in a document. This model is also famous for document classification tasks.

Python Implementation of the Naïve Bayes algorithm:

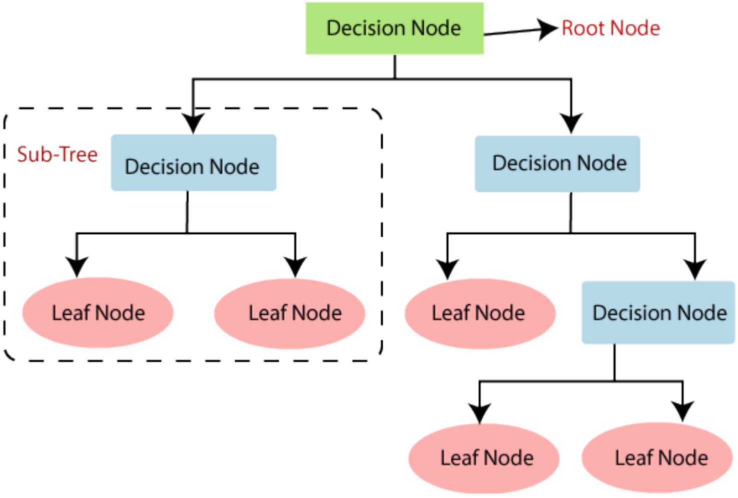
```
Importing the libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('user_data.csv')
x = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size =
0.25, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)

# Fitting Naive Bayes to the Training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(x_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(x_test)
```

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

# Visualising the Training set results
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
X1, X2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = 
x_set[:, 0].max() + 1, step = 0.01),
                     nm.arange(start = x_set[:, 1].min() - 1, stop = 
x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), 
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('purple', 'green')))
mtp.xlim(X1.min(), X1.max())
mtp.ylim(X2.min(), X2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
                c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Naive Bayes (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()

# Visualising the Test set results
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
X1, X2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = 
x_set[:, 0].max() + 1, step = 0.01),
                     nm.arange(start = x_set[:, 1].min() - 1, stop = 
x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), 
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('purple', 'green')))
mtp.xlim(X1.min(), X1.max())
mtp.ylim(X2.min(), X2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
                c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Naive Bayes (test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

4. Decision Tree Classification Algorithm

16 July 2023 19:09

1. Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.
2. In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
3. The decisions or the test are performed on the basis of features of the given dataset.
4. It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.
5. It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
6. In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm.
7. A decision tree can contain categorical data (YES/NO) as well as numeric data.



Why use Decision Trees?

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

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

Decision Tree Terminologies

Root Node: Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.

Leaf Node: Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
Splitting: Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.

Branch/Sub Tree: A tree formed by splitting the tree.
Pruning: Pruning is the process of removing the unwanted branches from the tree.

Parent/Child node: The root node of the tree is called the parent node, and other nodes are called the child nodes.

How does the Decision Tree algorithm Work?

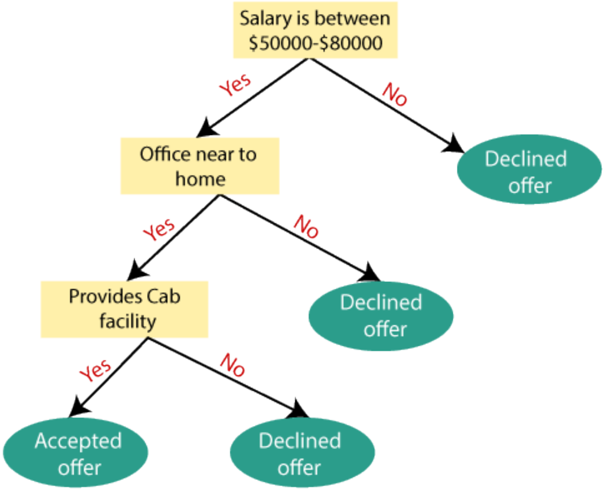
In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree.

The complete process can be better understood using the below algorithm:
Step-1: Begin the tree with the root node, says S, which contains the complete dataset.
Step-2: Find the best attribute in the dataset using Attribute Selection Measure (ASM).
Step-3: Divide the S into subsets that contains possible values for the best attributes.
Step-4: Generate the decision tree node, which contains the best attribute.
Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3.

- Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Example: Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not. So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM). The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels. The next decision node further gets split into one decision node (Cab facility) and one leaf node. Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer). Consider the below diagram:



Attribute Selection Measures

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

1. Information Gain
2. Gini Index

1. Information Gain:

1. Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.
2. It calculates how much information a feature provides us about a class.
3. According to the value of information gain, we split the node and build the decision tree.

4. A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:

   Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)

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

Entropy(s)= -P(yes)log2 P(yes)- P(no) log2 P(no)

Where,

S= Total number of samples
P(yes)= probability of yes
P(no)= probability of no

2. Gini Index:

Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.
An attribute with the low Gini index should be preferred as compared to the high Gini index.
It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.
Gini index can be calculated using the below formula:

Gini Index= 1- ∑jPj2

Pruning: Getting an Optimal Decision tree
Pruning is a process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree.

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

- Cost Complexity Pruning
- Reduced Error Pruning.

Advantages of the Decision Tree

1. It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
2. It can be very useful for solving decision-related problems.
3. It helps to think about all the possible outcomes for a problem.
4. There is less requirement of data cleaning compared to other algorithms.

Disadvantages of the Decision Tree

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

Python Implementation of Decision Tree

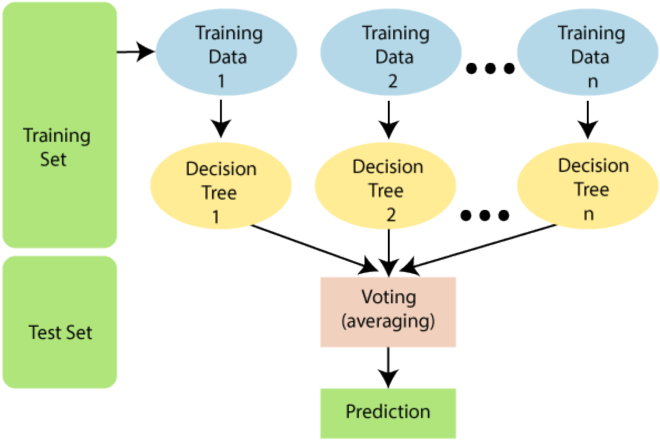
```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('user_data.csv')
```

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

# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25,
random_state=0)

#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)

#Fitting Decision Tree classifier to the training set
From sklearn.tree import DecisionTreeClassifier
classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
classifier.fit(x_train, y_train)

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

#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)

#Visulaizing the trianing set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop =
x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1,
step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
fori, j in enumerate(nm.unique(y_set)):
mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Decision Tree Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()

#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop =
x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1,
step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
fori, j in enumerate(nm.unique(y_set)):
mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Decision Tree Algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

5. Random Forest Algorithm

16 July 2023 19:30

1. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.
2. Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset.
3. The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.



Why use Random Forest?

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

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

How does Random Forest algorithm work?

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

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

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

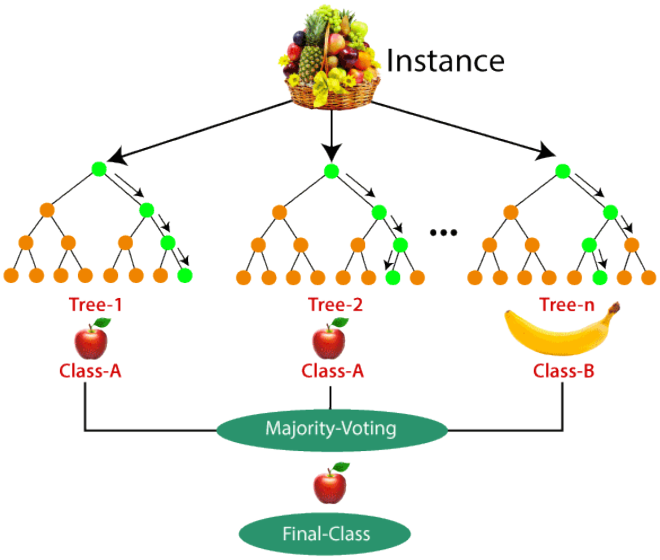
Step-2: Build the decision trees associated with the selected data points (Subsets).

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

Step-4: Repeat Step 1 & 2.

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

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



Applications of Random Forest

There are mainly four sectors where Random forest mostly used:

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

Advantages of Random Forest

1. Random Forest is capable of performing both Classification and Regression tasks.
2. It is capable of handling large datasets with high dimensionality.
3. It enhances the accuracy of the model and prevents the overfitting issue.
4. Disadvantages of Random Forest
5. Although random forest can be used for both classification and regression tasks, it is not more suitable for Regression tasks.

Python Implementation of Random Forest Algorithm

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('user_data.csv')

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

# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25,
random_state=0)

#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

```
#Fitting Decision Tree classifier to the training set
from sklearn.ensemble import RandomForestClassifier
classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")
classifier.fit(x_train, y_train)

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

#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)

from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop =
x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1,
step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Random Forest Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()

#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop =
x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1,
step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Random Forest Algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Unsupervised ML

08 July 2023 15:32

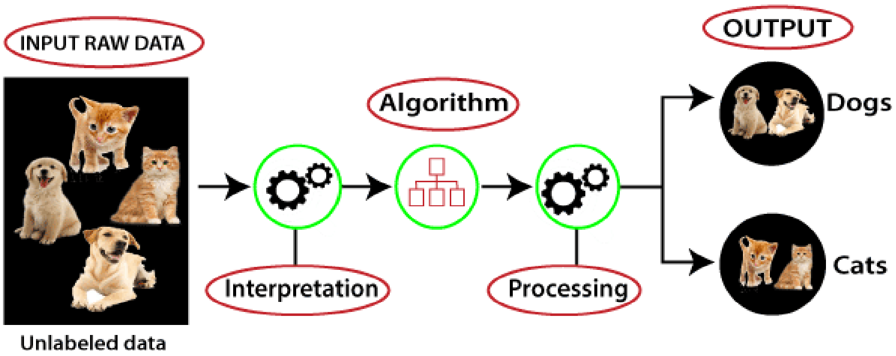
- Unsupervised learning is a type of machine learning in which models are trained using unlabelled dataset and are allowed to act on that data without any supervision.

- Unsupervised learning cannot be directly applied to a regression or classification problem because unlike supervised learning.

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

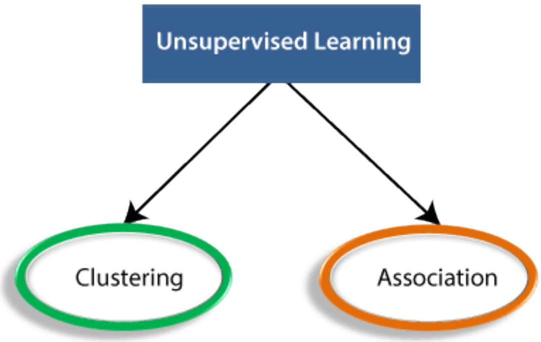
Working of Unsupervised Learning

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



Types of Unsupervised Learning Algorithm:

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



Clustering:

1. Clustering is a method of grouping the objects into clusters such that objects with most similarities remains into a group and has less or no similarities with the objects of another group.
2. Cluster analysis finds the commonalities between the data objects and categorizes them as per the presence and absence of those commonalities.

Association:

1. An association rule is an unsupervised learning method which is used for finding the relationships between variables in the large database. It determines the set of items that occurs together in the dataset.
2. Association rule makes marketing strategy more effective. Such as people who buy X item (suppose a bread) are also tend to purchase Y (Butter/Jam) item. A typical example of Association rule is Market Basket Analysis.

Unsupervised Learning algorithms:

Below is the list of some popular unsupervised learning algorithms:

1. *K-means clustering*
2. *KNN (k-nearest neighbours)*
3. *Hierarchal clustering*
4. *Anomaly detection*
5. *Neural Networks*
6. *Principle Component Analysis*
7. *Independent Component Analysis*
8. *Apriori algorithm*
9. *Singular value decomposition*

Clustering in Machine Learning

16 July 2023 19:42

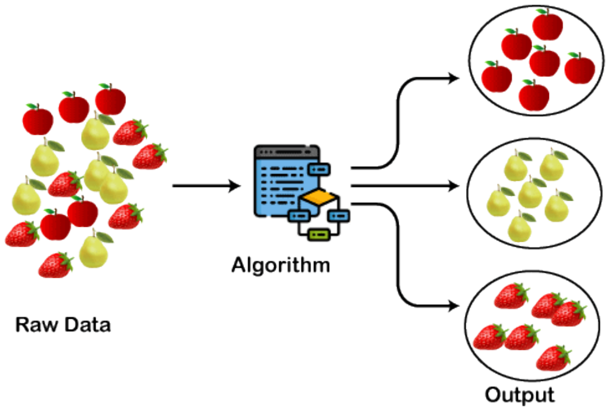
1. A way of grouping the data points into different clusters, consisting of similar data points. The objects with the possible similarities remain in a group that has less or no similarities with another group.
2. Clustering is somewhere similar to the classification algorithm, but the difference is the type of dataset that we are using. In classification, we work with the labelled data set, whereas in clustering, we work with the unlabelled dataset.
3. The clustering technique is commonly used for statistical data analysis.

Example: Let's understand the clustering technique with the real-world example of Mall: When we visit any shopping mall, we can observe that the things with similar usage are grouped together. Such as the t-shirts are grouped in one section, and trousers are at other sections, similarly, at vegetable sections, apples, bananas, Mangoes, etc., are grouped in separate sections, so that we can easily find out the things. The clustering technique also works in the same way. Other examples of clustering are grouping documents according to the topic.

The clustering technique can be widely used in various tasks. Some most common uses of this technique are:

1. Market Segmentation
2. Statistical data analysis
3. Social network analysis
4. Image segmentation
5. Anomaly detection, etc.

- *Apart from these general usages, it is used by the Amazon in its recommendation system to provide the recommendations as per the past search of products. Netflix also uses this technique to recommend the movies and web-series to its users as per the watch history.*



Types of Clustering Methods

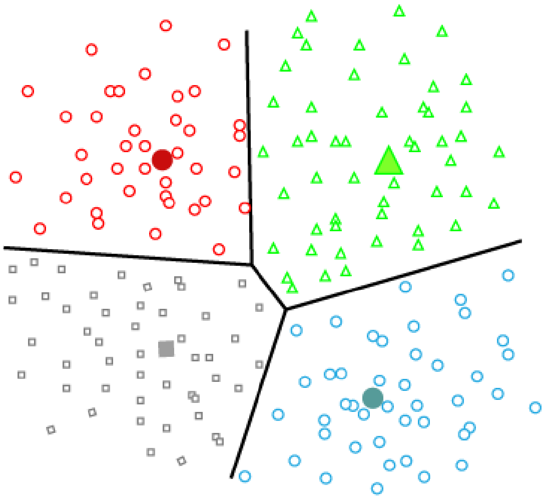
The clustering methods are broadly divided into Hard clustering (datapoint belongs to only one group) and Soft Clustering (data points can belong to another group also). But there are also other various approaches of Clustering exist. Below are the main clustering methods used in Machine learning:

1. Partitioning Clustering
2. Density-Based Clustering
3. Distribution Model-Based Clustering
4. Hierarchical Clustering
5. Fuzzy Clustering

Partitioning Clustering

It is a type of clustering that divides the data into non-hierarchical groups. It is also known as the **centroid-based method**. The most common example of partitioning clustering is the **K-Means Clustering algorithm**.

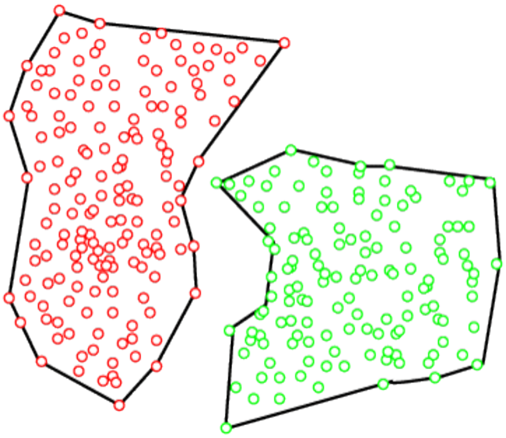
In this type, the dataset is divided into a set of k groups, where K is used to define the number of pre-defined groups. The cluster center is created in such a way that the distance between the data points of one cluster is minimum as compared to another cluster centroid.



Density-Based Clustering

The density-based clustering method connects the highly-dense areas into clusters, and the arbitrarily shaped distributions are formed as long as the dense region can be connected. This algorithm does it by identifying different clusters in the dataset and connects the areas of high densities into clusters. The dense areas in data space are divided from each other by sparser areas.

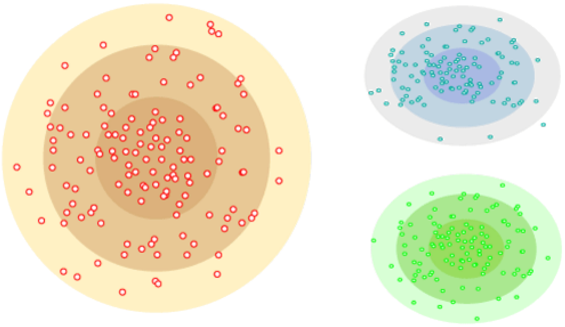
These algorithms can face difficulty in clustering the data points if the dataset has varying densities and high dimensions.



Distribution Model-Based Clustering

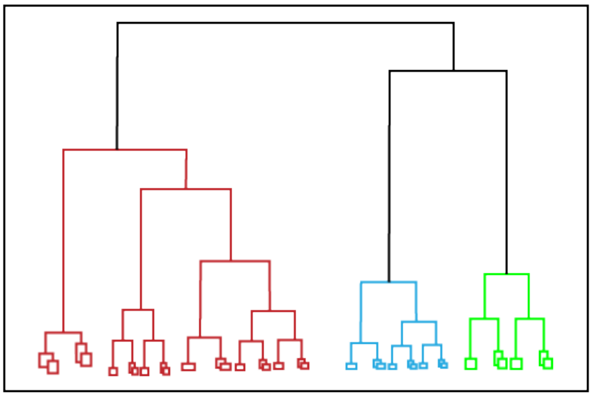
In the distribution model-based clustering method, the data is divided based on the probability of how a dataset belongs to a particular distribution. The grouping is done by assuming some distributions commonly **Gaussian Distribution**.

The example of this type is the **Expectation-Maximization Clustering algorithm** that uses Gaussian Mixture Models (GMM).



Hierarchical Clustering

Hierarchical clustering can be used as an alternative for the partitioned clustering as there is no requirement of pre-specifying the number of clusters to be created. In this technique, the dataset is divided into clusters to create a tree-like structure, which is also called a **dendrogram**. The observations or any number of clusters can be selected by cutting the tree at the correct level. The most common example of this method is the **Agglomerative Hierarchical algorithm**.



Fuzzy Clustering

Fuzzy clustering is a type of soft method in which a data object may belong to more than one group or cluster. Each dataset has a set of membership coefficients, which depend on the degree of membership to be in a cluster. **Fuzzy C-means algorithm** is the example of this type of clustering; it is sometimes also known as the Fuzzy k-means algorithm.

Clustering Algorithms

The Clustering algorithms can be divided based on their models that are explained above. There are different types of clustering algorithms published, but only a few are commonly used. The clustering algorithm is based on the kind of data that we are using. Such as, some algorithms need to guess the number of clusters in the given dataset, whereas some are required to find the minimum distance between the observation of the dataset.
Here we are discussing mainly popular Clustering algorithms that are widely used in machine learning:

1. **K-Means algorithm:** The k-means algorithm is one of the most popular clustering algorithms. It classifies the dataset by dividing the samples into different clusters of equal variances. The number of clusters must be specified in this algorithm. It is fast with fewer computations required, with the linear complexity of **O(n).**
2. **Mean-shift algorithm:** Mean-shift algorithm tries to find the dense areas in the smooth density of data points. It is an example of a centroid-based model, that works on updating the candidates for centroid to be the center of the points within a given region.
3. **DBSCAN Algorithm:** It stands **for Density-Based Spatial Clustering of Applications with Noise**. It is an example of a density-based model similar to the

mean-shift, but with some remarkable advantages. In this algorithm, the areas of high density are separated by the areas of low density. Because of this, the clusters can be found in any arbitrary shape.

4. **Expectation-Maximization Clustering using GMM:** This algorithm can be used as an alternative for the k-means algorithm or for those cases where K-means can be failed. In GMM, it is assumed that the data points are Gaussian distributed.
5. **Agglomerative Hierarchical algorithm:** The Agglomerative hierarchical algorithm performs the bottom-up hierarchical clustering. In this, each data point is treated as a single cluster at the outset and then successively merged. The cluster hierarchy can be represented as a tree-structure.
6. **Affinity Propagation:** It is different from other clustering algorithms as it does not require to specify the number of clusters. In this, each data point sends a message between the pair of data points until convergence. It has $O(N^2T)$ time complexity, which is the main drawback of this algorithm.

Applications of Clustering

Below are some commonly known applications of clustering technique in Machine Learning:

- **In Identification of Cancer Cells:** The clustering algorithms are widely used for the identification of cancerous cells. It divides the cancerous and non-cancerous data sets into different groups.
- **In Search Engines:** Search engines also work on the clustering technique. The search result appears based on the closest object to the search query. It does it by grouping similar data objects in one group that is far from the other dissimilar objects. The accurate result of a query depends on the quality of the clustering algorithm used.
- **Customer Segmentation:** It is used in market research to segment the customers based on their choice and preferences.
- **In Biology:** It is used in the biology stream to classify different species of plants and animals using the image recognition technique.
- **In Land Use:** The clustering technique is used in identifying the area of similar lands use in the GIS database. This can be very useful to find that for what purpose the particular land should be used, that means for which purpose it is more suitable.

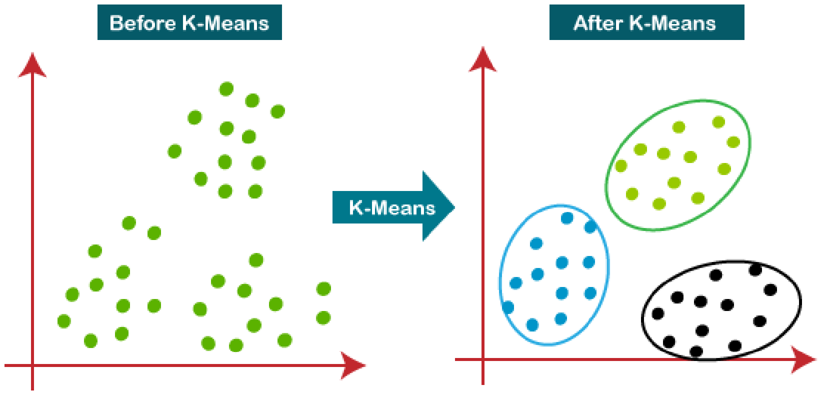
1.K-Means Clustering Algorithm

17 July 2023 16:40

1. It is an iterative algorithm that divides the unlabelled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.
2. It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabelled dataset on its own without the need for any training.
3. It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.
4. The algorithm takes the unlabelled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

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



How does the K-Means Algorithm Work?

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

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

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

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

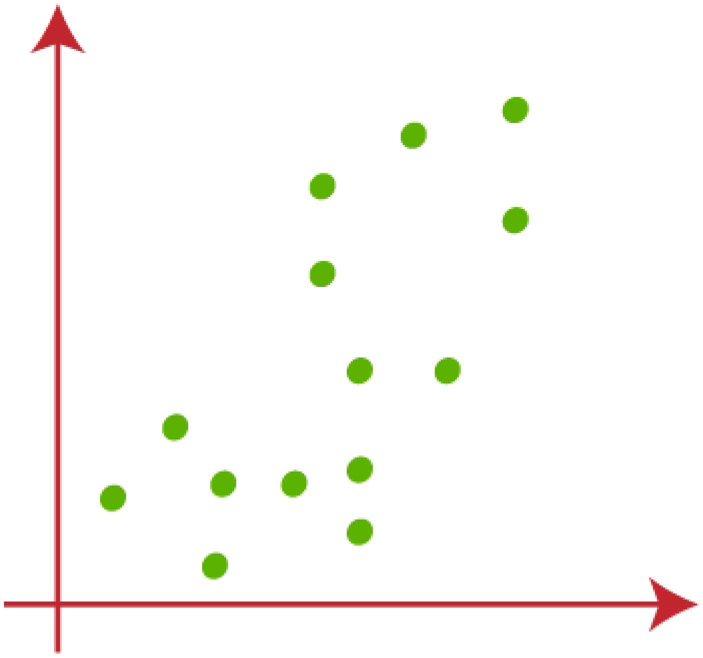
Step-4: Calculate the variance and place a new centroid of each cluster.

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

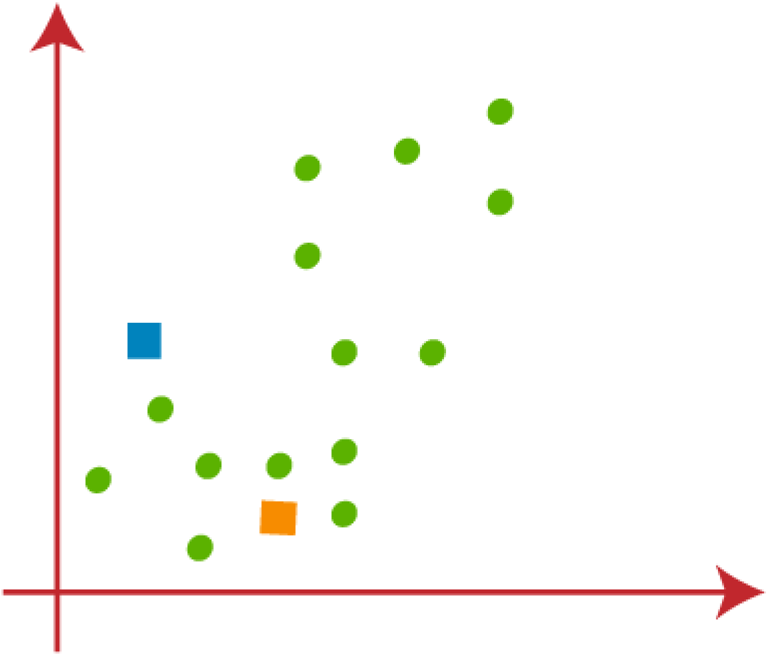
Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

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

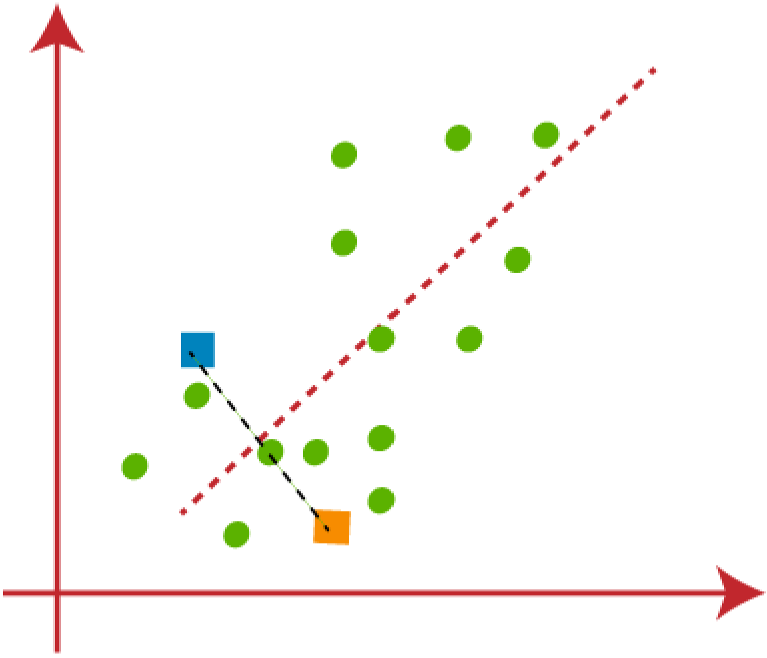


- Let's take number k of clusters, i.e., K=2, to identify the dataset and to put them into different clusters. It means here we will try to group these datasets into two different clusters.
- We need to choose some random k points or centroid to form the cluster. These points can be either the points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

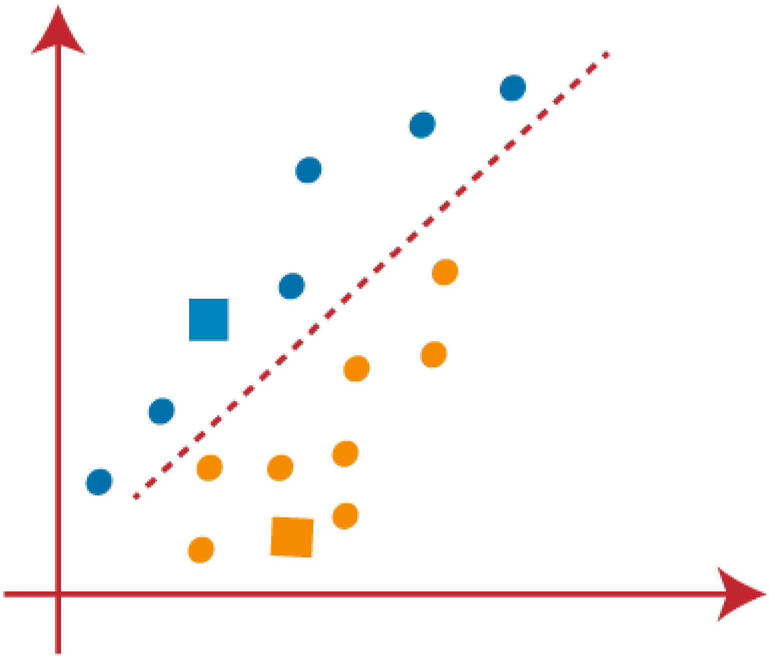


- Now we will assign each data point of the scatter plot to its closest K-point or centroid.

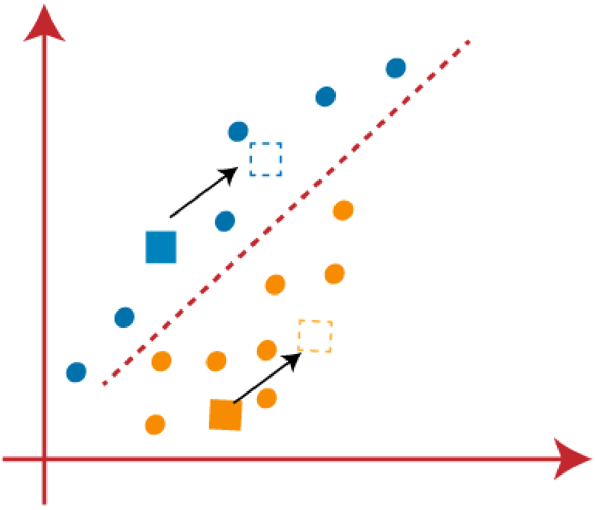
We will compute it by applying some mathematics that we have studied to calculate the distance between two points. So, we will draw a median between both the centroids. Consider the below image:



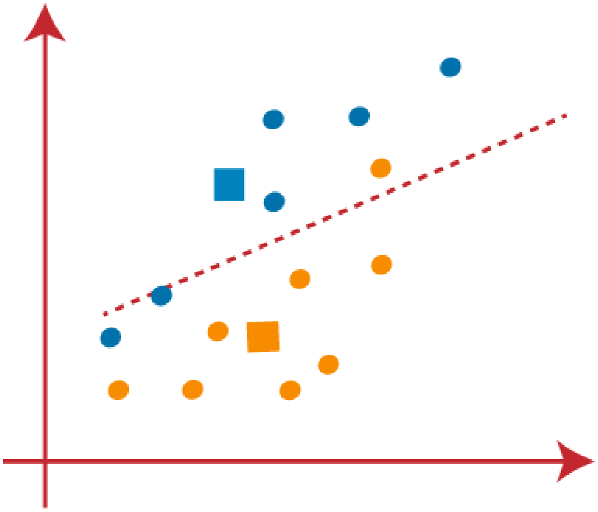
From the above image, it is clear that points left side of the line is near to the K1 or blue centroid, and points to the right of the line are close to the yellow centroid. Let's color them as blue and yellow for clear visualization.



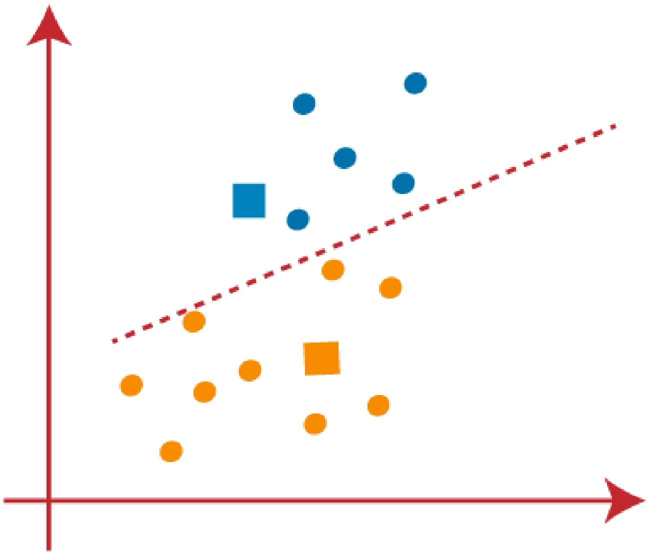
- As we need to find the closest cluster, so we will repeat the process by choosing **a new centroid**. To choose the new centroids, we will compute the center of gravity of these centroids, and will find new centroids as below:



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

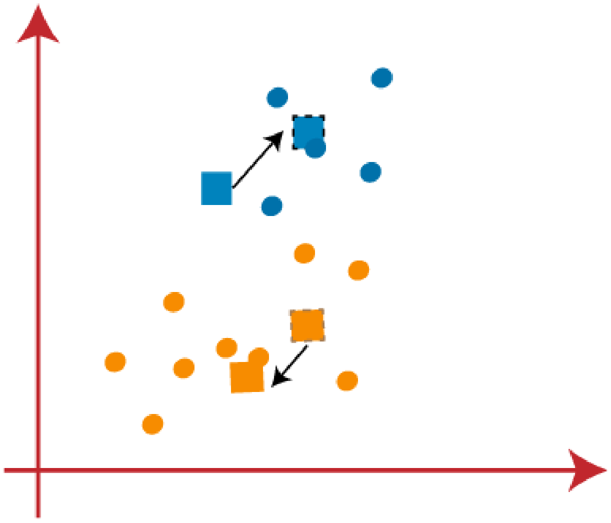


From the above image, we can see, one yellow point is on the left side of the line, and two blue points are right to the line. So, these three points will be assigned to new centroids.

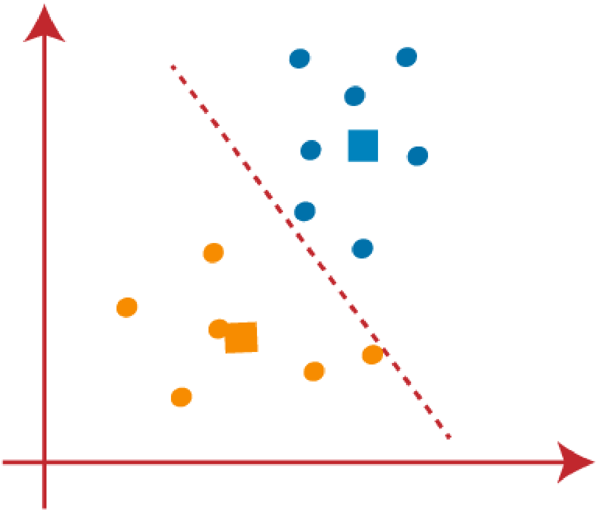


As reassignment has taken place, so we will again go to the step-4, which is finding new centroids or K-points.

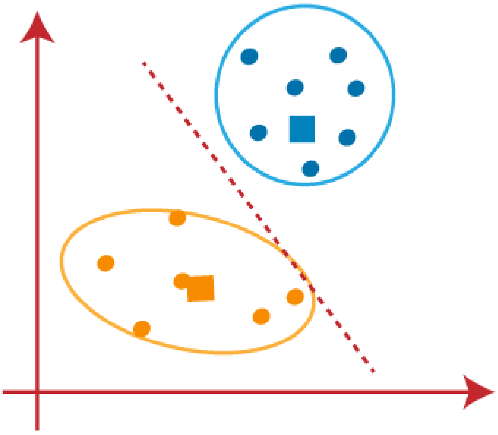
- We will repeat the process by finding the center of gravity of centroids, so the new centroids will be as shown in the below image:



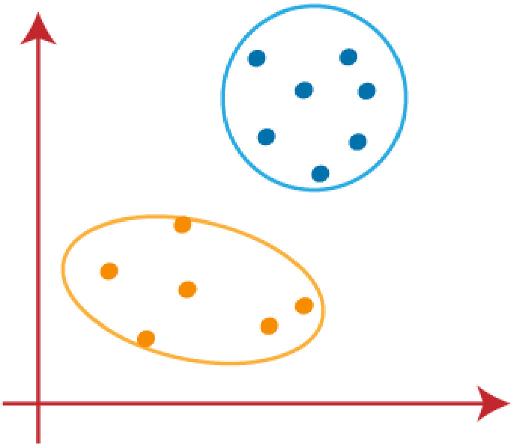
- As we got the new centroids so again will draw the median line and reassign the data points. So, the image will be:



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



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



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

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

Elbow Method

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

$$WCSS= \sum_{P_i \text{ in } Cluster1} distance(P_i\ C_1)^2 + \sum_{P_i \text{ in } Cluster2} distance(P_i\ C_2)^2 + \sum_{P_i \text{ in } CLuster3} distance(P_i\ C_3)^2$$

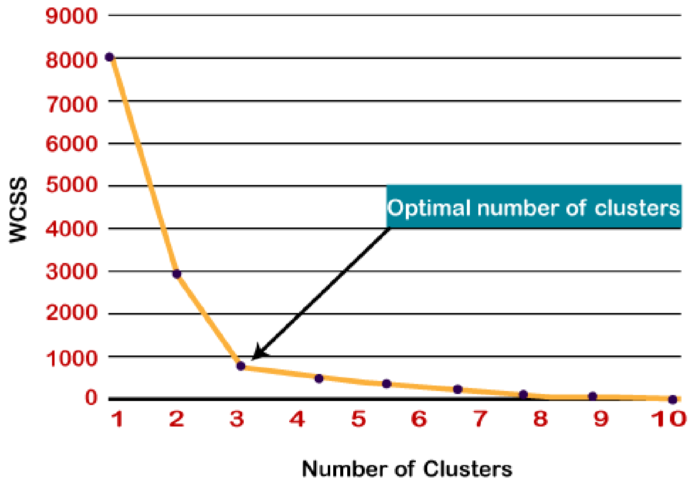
In the above formula of WCSS,
$\sum_{P_i \text{ in } Cluster1}$ distance$(P_i\ C_1)^2$: It is the sum of the square of the distances between each data point and its centroid within a cluster1 and the same for the other two terms.

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

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

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

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



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

Python Implementation of K-means Clustering Algorithm

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

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

x = dataset.iloc[:, [3, 4]].values

#finding optimal number of clusters using the elbow method
```

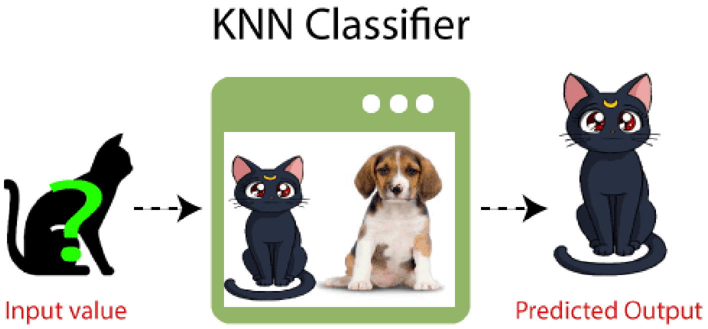
```
from sklearn.cluster import KMeans
wcss_list= []  #Initializing the list for the values of WCSS

#Using for loop for iterations from 1 to 10.
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state= 42)
    kmeans.fit(x)
    wcss_list.append(kmeans.inertia_)
mtp.plot(range(1, 11), wcss_list)
mtp.title('The Elobw Method Graph')
mtp.xlabel('Number of clusters(k)')
mtp.ylabel('wcss_list')
mtp.show()

#training the K-means model on a dataset
kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)
y_predict= kmeans.fit_predict(x)

#visulaizing the clusters
mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c =
'blue', label = 'Cluster 1') #for first cluster
mtp.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c =
'green', label = 'Cluster 2') #for second cluster
mtp.scatter(x[y_predict== 2, 0], x[y_predict == 2, 1], s = 100, c = 'red',
label = 'Cluster 3') #for third cluster
mtp.scatter(x[y_predict == 3, 0], x[y_predict == 3, 1], s = 100, c =
'cyan', label = 'Cluster 4') #for fourth cluster
mtp.scatter(x[y_predict == 4, 0], x[y_predict == 4, 1], s = 100, c =
'magenta', label = 'Cluster 5') #for fifth cluster
mtp.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
s = 300, c = 'yellow', label = 'Centroid')
mtp.title('Clusters of customers')
mtp.xlabel('Annual Income (k$)')
mtp.ylabel('Spending Score (1-100)')
mtp.legend()
mtp.show()
```

2. K-Nearest Neighbour(KNN)

16 July 2023 17:56

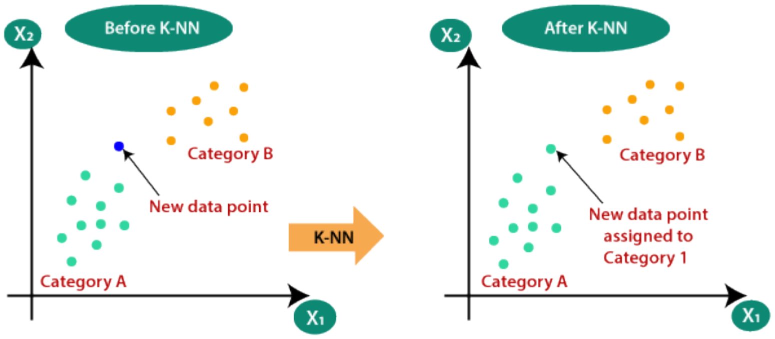
1. K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
2. K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
3. K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
4. K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.
5. It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
6. KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

Example: Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.



Why do we need a K-NN Algorithm?

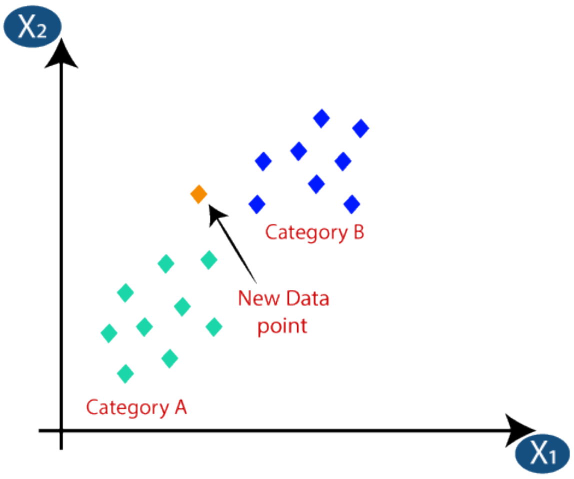
Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:



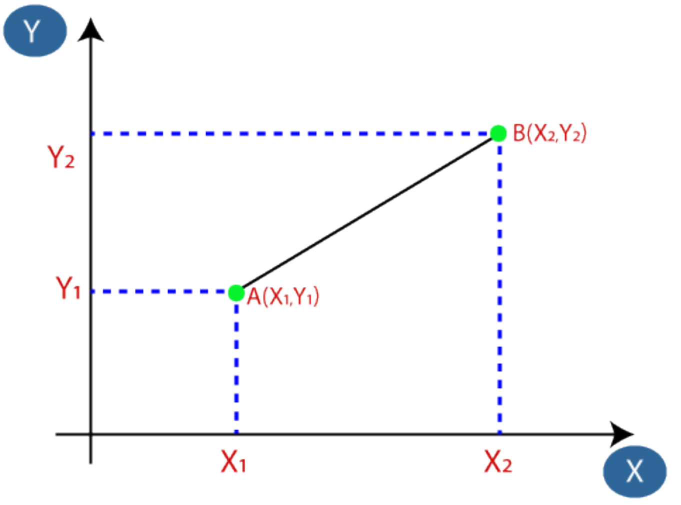
How does K-NN work?

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

Step-1: Select the number K of the neighbours
Step-2: Calculate the Euclidean distance of K number of neighbours
Step-3: Take the K nearest neighbours as per the calculated Euclidean distance.
Step-4: Among these k neighbours, count the number of the data points in each category.
Step-5: Assign the new data points to that category for which the number of the neighbour is maximum.
Step-6: Our model is ready.
Suppose we have a new data point and we need to put it in the required category. Consider the below image:



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



Euclidean Distance between A_1 and $B_2 = \sqrt{(X_2-X_1)^2+(Y_2-Y_1)^2}$

By calculating the Euclidean distance we got the nearest neighbours, as three nearest neighbours in category A and two nearest neighbours in category B. Consider the below image:
As we can see the 3 nearest neighbours are from category A, hence this new data point must belong to category A.

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

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

1. There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.

2. A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
3. Large values for K are good, but it may find some difficulties.
4. Advantages of KNN Algorithm:
5. It is simple to implement.
6. It is robust to the noisy training data
7. It can be more effective if the training data is large.
8. Disadvantages of KNN Algorithm:
9. Always needs to determine the value of K which may be complex some time.
10. The computation cost is high because of calculating the distance between the data points for all the training samples.

Advantages of KNN Algorithm:

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

Disadvantages of KNN Algorithm:

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

Python implementation of the KNN algorithm

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

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('user_data.csv')

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

# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25,
random_state=0)

#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)

#Fitting K-NN classifier to the training set
from sklearn.neighbors import KNeighborsClassifier
classifier= KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2 )
classifier.fit(x_train, y_train)
```

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

#Creating the Confusion matrix
    from sklearn.metrics import confusion_matrix
    cm= confusion_matrix(y_test, y_pred)

#Visulaizing the trianing set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop =
x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1,
step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('K-NN Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()

#Visualizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop =
x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1,
step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('K-NN algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

3. Hierarchical Clustering in Machine Learning

16 July 2023 19:58

Hierarchical clustering is another unsupervised machine learning algorithm, which is used to group the unlabelled datasets into a cluster and also known as hierarchical cluster analysis or HCA.

The hierarchical clustering technique has two approaches:

1. **Agglomerative:** Agglomerative is a bottom-up approach, in which the algorithm starts with taking all data points as single clusters and merging them until one cluster is left.
2. **Divisive:** Divisive algorithm is the reverse of the agglomerative algorithm as it is a top-down approach.

Agglomerative Hierarchical clustering

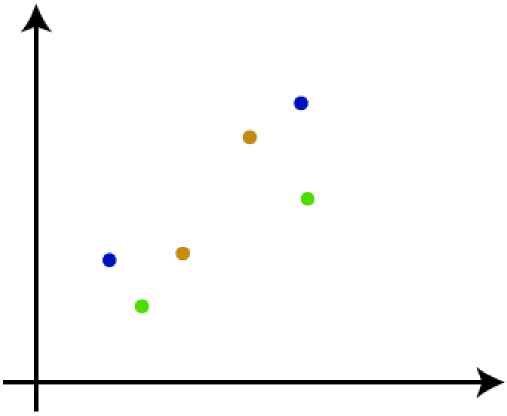
The agglomerative hierarchical clustering algorithm is a popular example of HCA. To group the datasets into clusters, it follows the **bottom-up approach**. It means, this algorithm considers each dataset as a single cluster at the beginning, and then start combining the closest pair of clusters together. It does this until all the clusters are merged into a single cluster that contains all the datasets.

This hierarchy of clusters is represented in the form of the dendrogram.

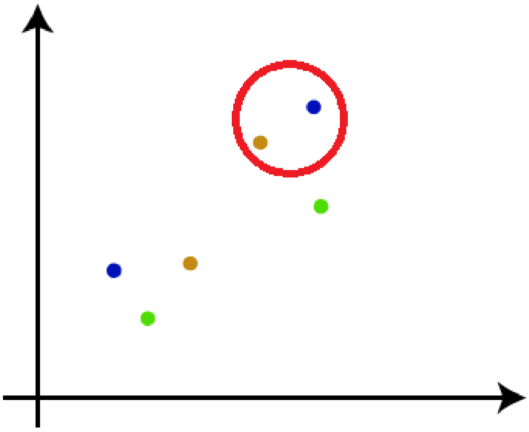
How the Agglomerative Hierarchical clustering Work?

The working of the AHC algorithm can be explained using the below steps:

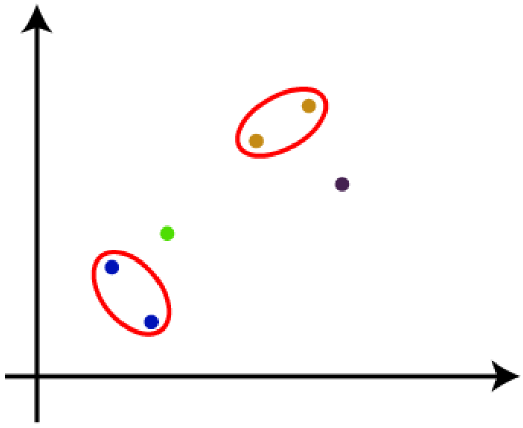
- **Step-1:** Create each data point as a single cluster. Let's say there are N data points, so the number of clusters will also be N.



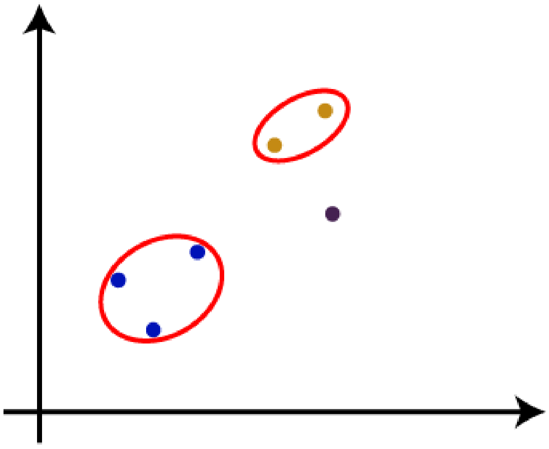
- **Step-2:** Take two closest data points or clusters and merge them to form one cluster. So, there will now be N-1 clusters.

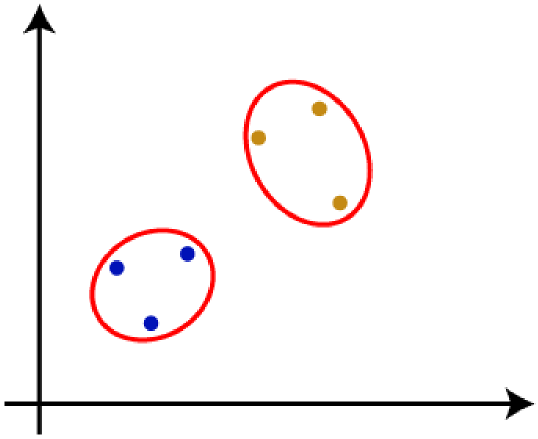


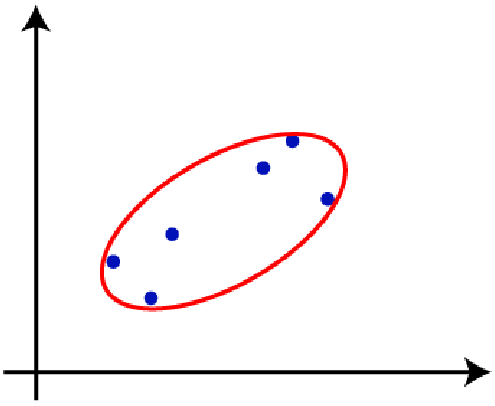
- **Step-3**: Again, take the two closest clusters and merge them together to form one cluster. There will be N-2 clusters.



- **Step-4:** Repeat Step 3 until only one cluster left. So, we will get the following clusters. Consider the below images:







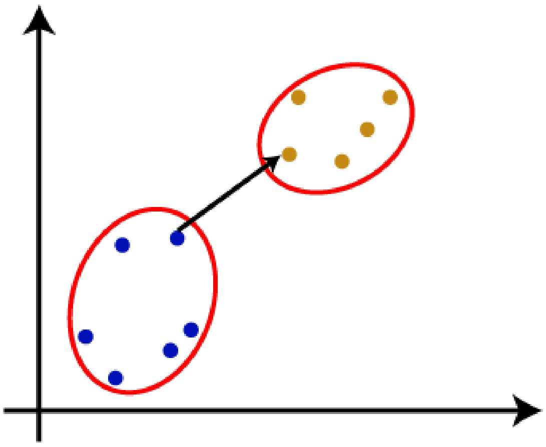
- **Step-5:** Once all the clusters are combined into one big cluster, develop the dendrogram to divide the clusters as per the problem.

Note: To better understand hierarchical clustering, it is advised to have a look on k-means clustering

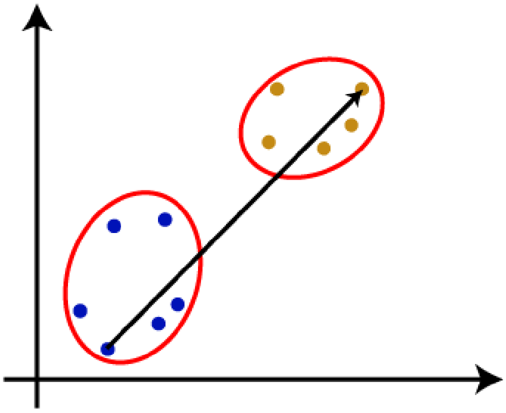
Measure for the distance between two clusters

As we have seen, the **closest distance** between the two clusters is crucial for the hierarchical clustering. There are various ways to calculate the distance between two clusters, and these ways decide the rule for clustering. These measures are called **Linkage methods**. Some of the popular linkage methods are given below:

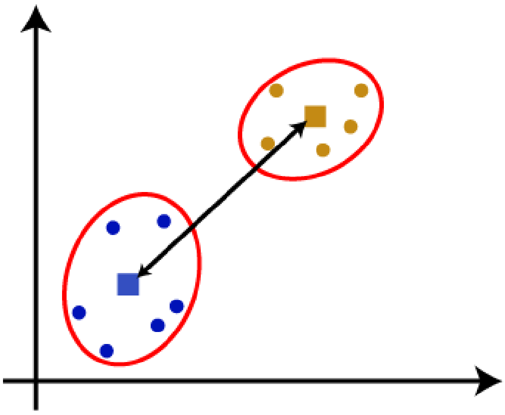
1. **Single Linkage:** It is the Shortest Distance between the closest points of the clusters. Consider the below image:



2. **Complete Linkage:** It is the farthest distance between the two points of two different clusters. It is one of the popular linkage methods as it forms tighter clusters than single-linkage.



3. **Average Linkage:** It is the linkage method in which the distance between each pair of datasets is added up and then divided by the total number of datasets to calculate the average distance between two clusters. It is also one of the most popular linkage methods.
4. **Centroid Linkage:** It is the linkage method in which the distance between the centroid of the clusters is calculated. Consider the below image:

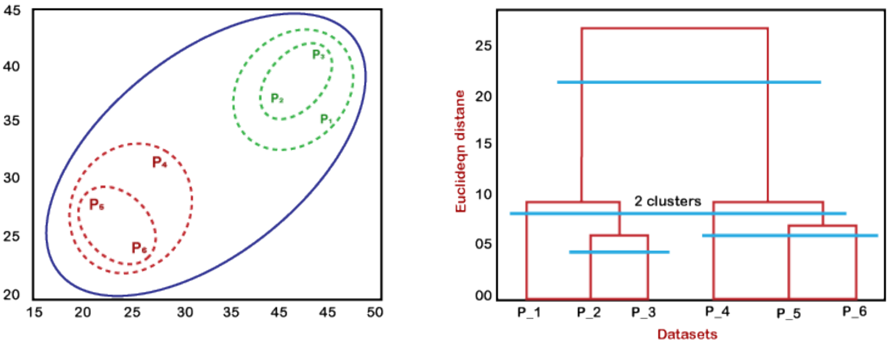


From the above-given approaches, we can apply any of them according to the type of problem or business requirement.

Woking of Dendrogram in Hierarchical clustering

The dendrogram is a tree-like structure that is mainly used to store each step as a memory that the HC algorithm performs. In the dendrogram plot, the Y-axis shows the Euclidean distances between the data points, and the x-axis shows all the data points of the given dataset.

The working of the dendrogram can be explained using the below diagram:



In the above diagram, the left part is showing how clusters are created in agglomerative clustering, and the right part is showing the corresponding dendrogram.

- As we have discussed above, firstly, the datapoints P2 and P3 combine together and form a cluster, correspondingly a dendrogram is created, which connects P2 and P3 with a rectangular shape. The hight is decided according to the Euclidean distance between the data points.
- In the next step, P5 and P6 form a cluster, and the corresponding dendrogram is created. It is higher than of previous, as the Euclidean distance between P5 and P6 is a little bit greater than the P2 and P3.
- Again, two new dendrograms are created that combine P1, P2, and P3 in one dendrogram, and P4, P5, and P6, in another dendrogram.
- At last, the final dendrogram is created that combines all the data points together.

Python Implementation of Agglomerative Hierarchical Clustering

```
# Importing the libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

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

x = dataset.iloc[:, [3, 4]].values

#Finding the optimal number of clusters using the dendrogram
import scipy.cluster.hierarchy as shc
dendro = shc.dendrogram(shc.linkage(x, method="ward"))
mtp.title("Dendrogrma Plot")
mtp.ylabel("Euclidean Distances")
mtp.xlabel("Customers")
mtp.show()

#training the hierarchical model on dataset
from sklearn.cluster import AgglomerativeClustering
hc= AgglomerativeClustering(n_clusters=5, affinity='euclidean',
linkage='ward')
y_pred= hc.fit_predict(x)

#visulaizing the clusters
mtp.scatter(x[y_pred == 0, 0], x[y_pred == 0, 1], s = 100, c = 'blue',
label = 'Cluster 1')
mtp.scatter(x[y_pred == 1, 0], x[y_pred == 1, 1], s = 100, c = 'green',
label = 'Cluster 2')
mtp.scatter(x[y_pred== 2, 0], x[y_pred == 2, 1], s = 100, c = 'red', label
= 'Cluster 3')
mtp.scatter(x[y_pred == 3, 0], x[y_pred == 3, 1], s = 100, c = 'cyan',
label = 'Cluster 4')
mtp.scatter(x[y_pred == 4, 0], x[y_pred == 4, 1], s = 100, c = 'magenta',
label = 'Cluster 5')
mtp.title('Clusters of customers')
mtp.xlabel('Annual Income (k$)')
mtp.ylabel('Spending Score (1-100)')
mtp.legend()
mtp.show()
```

Association in Machine Learning

17 July 2023 16:50

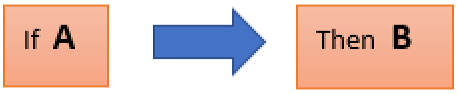
1. The association rule learning is one of the very important concepts of machine learning, and it is employed in Market Basket analysis, Web usage mining, continuous production, etc.
2. Here market basket analysis is a technique used by the various big retailer to discover the associations between items.
3. We can understand it by taking an example of a supermarket, as in a supermarket, all products that are purchased together are put together.

Association rule learning can be divided into three types of algorithms:

1. *Apriori*
2. *Eclat*
3. *F-P Growth Algorithm*

How does Association Rule Learning work?

Association rule learning works on the concept of If and Else Statement, such as if A then B.



Here the If element is called **antecedent**, and then statement is called as **Consequent**. These types of relationships where we can find out some association or relation between two items is known *as single cardinality*. It is all about creating rules, and if the number of items increases, then cardinality also increases accordingly. So, to measure the associations between thousands of data items, there are several metrics. These metrics are given below:

- **Support**
- **Confidence**
- **Lift**

Let's understand each of them:

Support

Support is the frequency of A or how frequently an item appears in the dataset. It is defined as the fraction of the transaction T that contains the itemset X. If there are X datasets, then for transactions T, it can be written as:

$$\text{Supp(X)=}\frac{Freq(X)}{T}$$

Confidence

Confidence indicates how often the rule has been found to be true. Or how often the items X and Y occur together in the dataset when the occurrence of X is already given. It is the ratio of the transaction that contains X and Y to the number of records that contain X.

$$\text{Confidence=}\frac{Freq(X,Y)}{Freq(X)}$$

Lift

It is the strength of any rule, which can be defined as below formula:

$$\text{Lift=}\frac{Supp(X,Y)}{Supp(X)\times Supp(Y)}$$

It is the ratio of the observed support measure and expected support if X and Y are independent of each other. It has three possible values:

- If **Lift= 1**: The probability of occurrence of antecedent and consequent is independent of each other.
- **Lift>1**: It determines the degree to which the two itemsets are dependent to each other.
- **Lift<1**: It tells us that one item is a substitute for other items, which means one item has a negative effect on another.

Types of Association Rule Learning

Association rule learning can be divided into three algorithms:

Apriori Algorithm

This algorithm uses frequent datasets to generate association rules. It is designed to work on the databases that contain transactions. This algorithm uses a breadth-first search and Hash Tree to calculate the itemset efficiently.
It is mainly used for market basket analysis and helps to understand the products that can be bought together. It can also be used in the healthcare field to find drug reactions for patients.

Eclat Algorithm

Eclat algorithm stands for **Equivalence Class Transformation**. This algorithm uses a depth-first search technique to find frequent itemsets in a transaction database. It performs faster execution than Apriori Algorithm.

F-P Growth Algorithm

The F-P growth algorithm stands for **Frequent Pattern**, and it is the improved version of the Apriori Algorithm. It represents the database in the form of a tree structure that is known as a frequent pattern or tree. The purpose of this frequent tree is to extract the most frequent patterns.

Applications of Association Rule Learning

It has various applications in machine learning and data mining. Below are some popular applications of association rule learning:

- **Market Basket Analysis:** It is one of the popular examples and applications of association rule mining. This technique is commonly used by big retailers to determine the association between items.
- **Medical Diagnosis:** With the help of association rules, patients can be cured easily, as it helps in identifying the probability of illness for a particular disease.
- **Protein Sequence:** The association rules help in determining the synthesis of artificial Proteins.
- It is also used for the **Catalogue Design** and **Loss-leader Analysis** and many more other applications.

Apriori Algorithm in Machine Learning

18 July 2023 16:53

1. The Apriori algorithm uses frequent itemsets to generate association rules, and it is designed to work on the databases that contain transactions.
2. With the help of these association rule, it determines how strongly or how weakly two objects are connected.
3. This algorithm uses a breadth-first search and Hash Tree to calculate the itemset associations efficiently.
4. It is the iterative process for finding the frequent itemsets from the large dataset.

★ This algorithm was given by the R. Agrawal and Srikant in the year 1994. It is mainly used for market basket analysis and helps to find those products that can be bought together. It can also be used in the healthcare field to find drug reactions for patients.

What is Frequent Itemset?

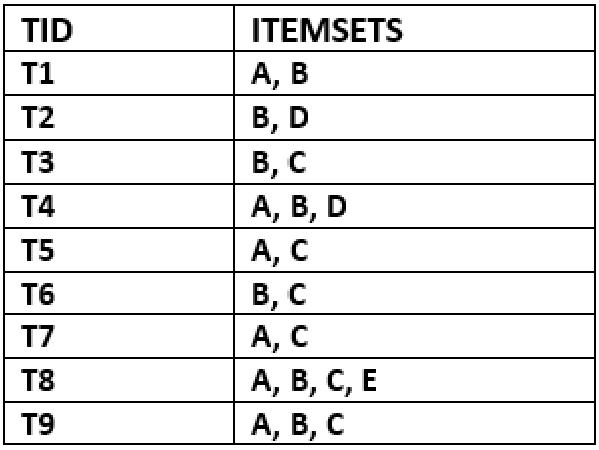
Frequent itemsets are those items whose support is greater than the threshold value or user-specified minimum support. It means if A & B are the frequent itemsets together, then individually A and B should also be the frequent itemset. Suppose there are the two transactions: A= {1,2,3,4,5}, and B= {2,3,7}, in these two transactions, 2 and 3 are the frequent itemsets.

Steps for Apriori Algorithm

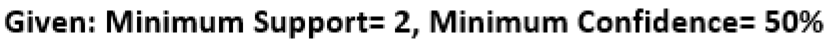
Below are the steps for the apriori algorithm:
Step-1: Determine the support of itemsets in the transactional database, and select the minimum support and confidence.
Step-2: Take all supports in the transaction with higher support value than the minimum or selected support value.
Step-3: Find all the rules of these subsets that have higher confidence value than the threshold or minimum confidence.
Step-4: Sort the rules as the decreasing order of lift.

Apriori Algorithm Working

We will understand the apriori algorithm using an example and mathematical calculation:
Example: Suppose we have the following dataset that has various transactions, and from this dataset, we need to find the frequent itemsets and generate the association rules using the Apriori algorithm:



TID	ITEMSETS
T1	A, B
T2	B, D
T3	B, C
T4	A, B, D
T5	A, C
T6	B, C
T7	A, C
T8	A, B, C, E
T9	A, B, C

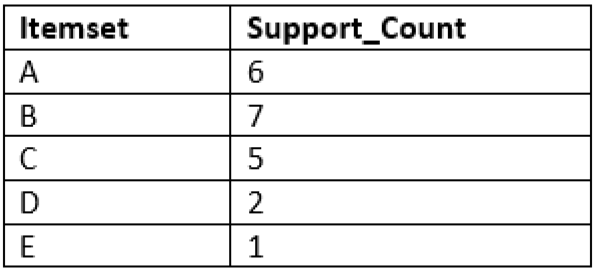


Given: Minimum Support= 2, Minimum Confidence= 50%

Solution:

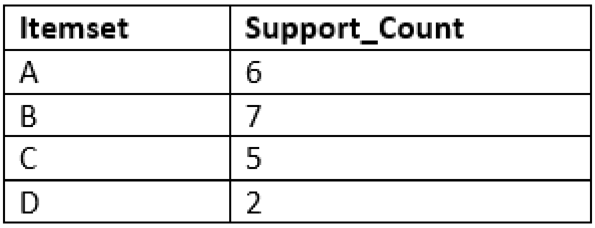
Step-1: Calculating C1 and L1:

- In the first step, we will create a table that contains support count (The frequency of each itemset individually in the dataset) of each itemset in the given dataset. This table is called the **Candidate set or C1.**



Itemset	Support_Count
A	6
B	7
C	5
D	2
E	1

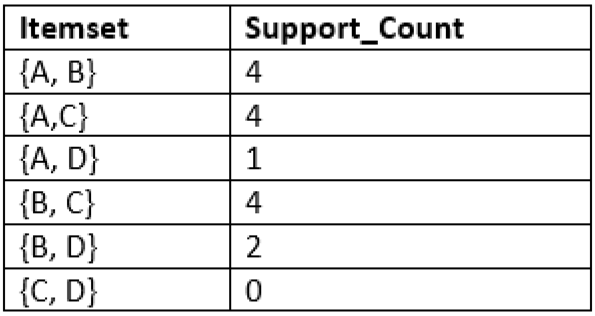
- Now, we will take out all the itemsets that have the greater support count that the Minimum Support (2). It will give us the table for the **frequent itemset L1.**
  Since all the itemsets have greater or equal support count than the minimum support, except the E, so E itemset will be removed.



Itemset	Support_Count
A	6
B	7
C	5
D	2

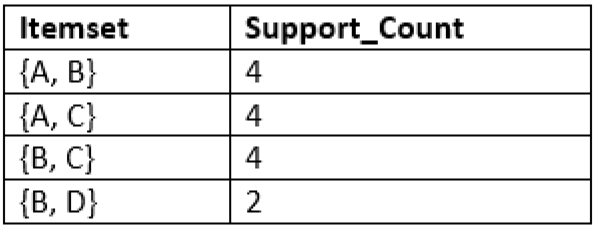
Step-2: Candidate Generation C2, and L2:

- In this step, we will generate C2 with the help of L1. In C2, we will create the pair of the itemsets of L1 in the form of subsets.
- After creating the subsets, we will again find the support count from the main transaction table of datasets, i.e., how many times these pairs have occurred together in the given dataset. So, we will get the below table for C2:



Itemset	Support_Count
{A, B}	4
{A,C}	4
{A, D}	1
{B, C}	4
{B, D}	2
{C, D}	0

- Again, we need to compare the C2 Support count with the minimum support count, and after comparing, the itemset with less support count will be eliminated from the table C2. It will give us the below table for L2

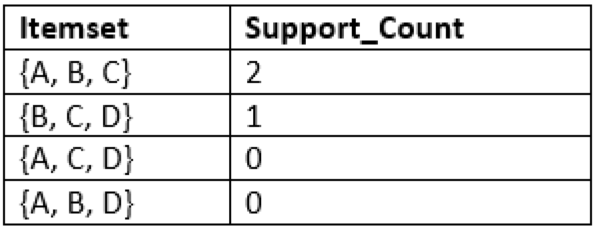


Itemset	Support_Count
{A, B}	4
{A, C}	4
{B, C}	4
{B, D}	2

A, B, C, D

Step-3: Candidate generation C3, and L3:

- For C3, we will repeat the same two processes, but now we will form the C3 table with subsets of three itemsets together, and will calculate the support count from the dataset. It will give the below table:



Itemset	Support_Count
{A, B, C}	2
{B, C, D}	1
{A, C, D}	0
{A, B, D}	0

- Now we will create the L3 table. As we can see from the above C3 table, there is only one combination of itemset that has support count equal to the minimum support count. So, the L3 will have only one combination, i.e., **{A, B, C}.**

Step-4: Finding the association rules for the subsets:

To generate the association rules, first, we will create a new table with the possible rules from the occurred combination {A, B.C}. For all the rules, we will calculate the Confidence using formula **sup(A ^B)/A.** After calculating the confidence value for all rules, we will exclude the rules that have less confidence than the minimum threshold(50%).
Consider the below table:

Rules	Support	Confidence
A ^B → C	2	Sup{(A ^B) ^C}/sup(A ^B)= 2/4=0.5=50%
B^C → A	2	Sup{(B^C) ^A}/sup(B ^C)= 2/4=0.5=50%
A^C → B	2	Sup{(A ^C) ^B}/sup(A ^C)= 2/4=0.5=50%
C→ A ^B	2	Sup{(C^(A ^B)}/sup(C)= 2/5=0.4=40%
A→ B^C	2	Sup{(A^(B ^C)}/sup(A)= 2/6=0.33=33.33%
B→ B^C	2	Sup{(B^(B ^C)}/sup(B)= 2/7=0.28=28%

As the given threshold or minimum confidence is 50%, so the first three rules **A ^B → C, B^C → A, and A^C → B** can be considered as the strong association rules for the given problem.

Advantages of Apriori Algorithm

- This is easy to understand algorithm
- The join and prune steps of the algorithm can be easily implemented on large datasets.

Disadvantages of Apriori Algorithm

- The apriori algorithm works slow compared to other algorithms.
- The overall performance can be reduced as it scans the database for multiple times.

- The time complexity and space complexity of the apriori algorithm is $O(2^D)$, which is very high. Here D represents the horizontal width present in the database.

Python Implementation of Apriori Algorithm

→ pip install apyroi

```
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#Importing the dataset
dataset = pd.read_csv('Market_Basket_data1.csv')
transactions=[]
for i in range(0, 7501):
    transactions.append([str(dataset.values[i,j])  for j in range(0,20)])

from apyori import apriori
rules= apriori(transactions= transactions, min_support=0.003,
min_confidence = 0.2, min_lift=3, min_length=2, max_length

results= list(rules)
results

for item in results:
    pair = item[0]
    items = [x for x in pair]
    print("Rule: " + items[0] + " -> " + items[1])

    print("Support: " + str(item[1]))
    print("Confidence: " + str(item[2][0][2]))
    print("Lift: " + str(item[2][0][3]))
    print("=====================================")
```

Supervised vs Unsupervised

08 July 2023 15:50

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

Regression Analysis

08 July 2023 15:52

1. Regression analysis is a statistical method to model the relationship between a dependent (target) and independent (predictor) variables with one or more independent variables.
2. 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.
3. It predicts continuous/real values such as **temperature, age, salary, price,** etc.

Regression shows a line or curve that passes through all the datapoints on target-predictor graph in such a way that the vertical distance between the datapoints and the regression line is minimum.

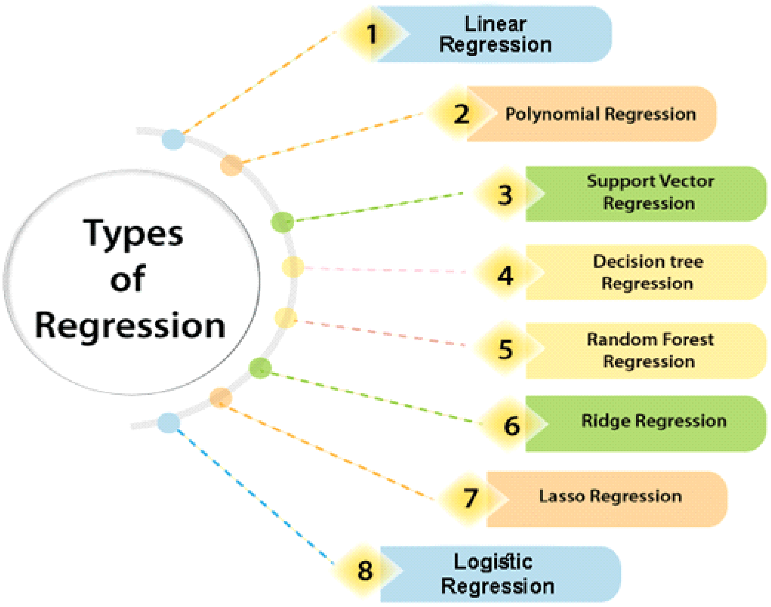
Terminologies Related to the Regression Analysis:

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

Types of Regression:

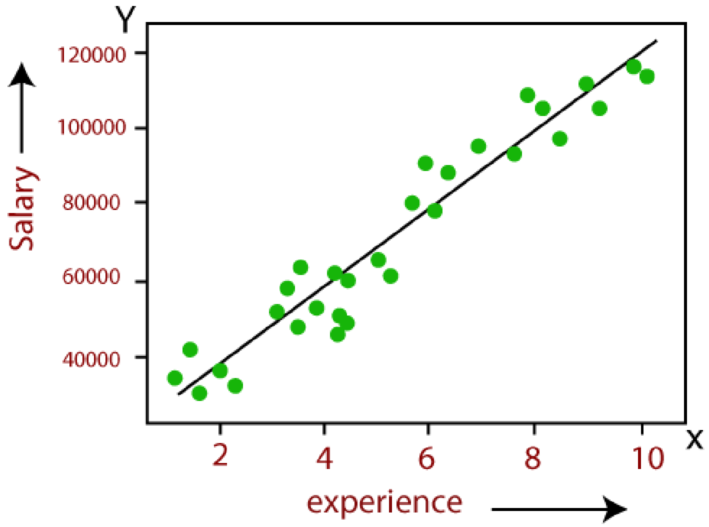
There are various types of regressions which are used in data science and machine learning. Each type has its own importance on different scenarios, but at the core, all the regression methods analyse the effect of the independent variable on dependent variables. Here we are discussing some important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

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



Linear Regression:

- Linear regression is a statistical regression method which is used for predictive analysis.
- It is one of the very simple and easy algorithms which works on regression and shows the relationship between the continuous variables.
- It is used for solving the regression problem in machine learning.
- Linear regression shows the linear relationship between the independent variable (X-axis) and the dependent variable (Y-axis), hence called linear regression.
- If there is only one input variable (x), then such linear regression is called **simple linear regression**. And if there is more than one input variable, then such linear regression is called **multiple linear regression**.
- The relationship between variables in the linear regression model can be explained using the below image. Here we are predicting the salary of an employee on the basis of **the year of experience**.



<u>Below is the mathematical equation for Linear regression:</u>

Y= aX+b
Y = dependent variables (target variables),
X= Independent variables (predictor variables),
a and b are the linear coefficients

Some popular applications of linear regression are:

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

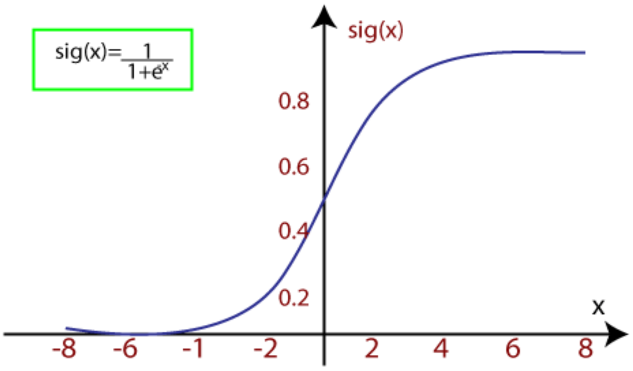
Logistic Regression:

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

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

- f(x)= Output between the 0 and 1 value.
- x= input to the function
- e= base of natural logarithm.

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



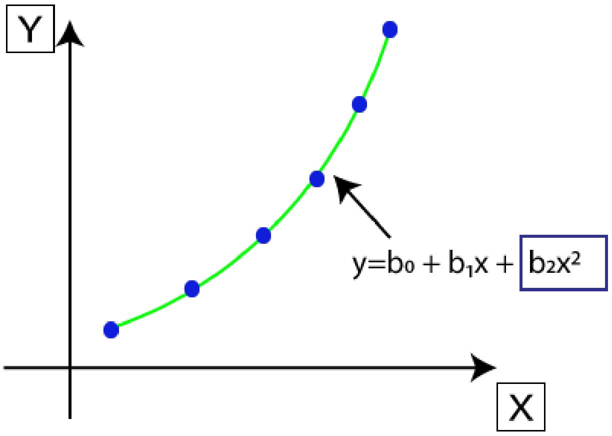
- It uses the concept of threshold levels, values above the threshold level are rounded up to 1, and values below the threshold level are rounded up to 0.

There are three types of logistic regression:

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

Polynomial Regression:

- Polynomial Regression is a type of regression which models the **non-linear dataset** using a linear model.
- It is similar to multiple linear regression, but it fits a non-linear curve between the value of x and corresponding conditional values of y.
- Suppose there is a dataset which consists of datapoints which are present in a non-linear fashion, so for such case, linear regression will not best fit to those datapoints. To cover such datapoints, we need Polynomial regression.
- In **Polynomial regression, the original features are transformed into polynomial features of given degree and then modeled using a linear model.** Which means the datapoints are best fitted using a polynomial line.



- The equation for polynomial regression also derived from linear regression equation that means Linear regression equation $Y= b_0+ b_1x$, is transformed into Polynomial regression equation $Y= b_0+b_1x+ b_2x^2+ b_3x^3+.....+ b_nx^n$.
- Here Y is the **predicted/target output, b_0, b_1,... b_n are the regression coefficients**. x is our **independent/input variable**.
- The model is still linear as the coefficients are still linear with quadratic

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

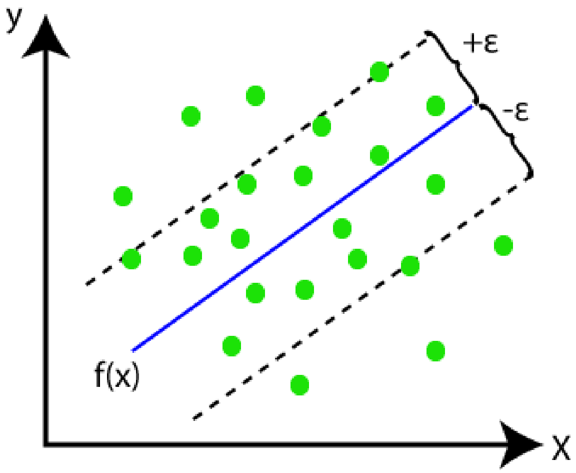
Support Vector Regression:

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

Vector Regression.
Support Vector Regression is a regression algorithm which works for continuous variables. Below are some keywords which are used in **Support Vector Regression**:

- **Kernel:** It is a function used to map a lower-dimensional data into higher dimensional data.
- **Hyperplane:** In general SVM, it is a separation line between two classes, but in SVR, it is a line which helps to predict the continuous variables and cover most of the datapoints.
- **Boundary line:** Boundary lines are the two lines apart from hyperplane, which creates a margin for datapoints.
- **Support vectors:** Support vectors are the datapoints which are nearest to the hyperplane and opposite class.

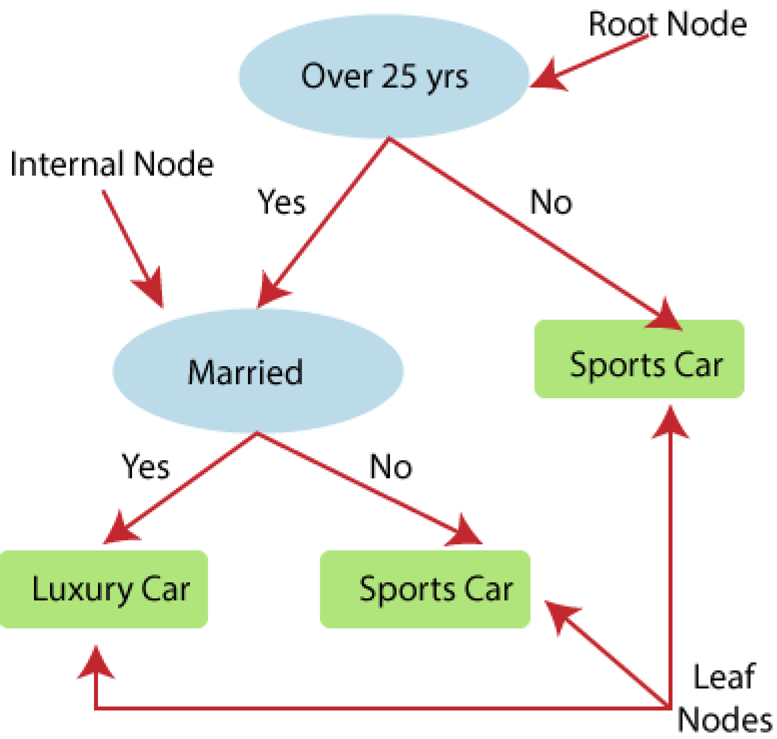
In SVR, we always try to determine a hyperplane with a maximum margin, so that maximum number of datapoints are covered in that margin. ***The main goal of SVR is to consider the maximum datapoints within the boundary lines and the hyperplane (best-fit line) must contain a maximum number of datapoints***.



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

Decision Tree Regression:

- Decision Tree is a supervised learning algorithm which can be used for solving both classification and regression problems.
- It can solve problems for both categorical and numerical data
- Decision Tree regression builds a tree-like structure in which each internal node represents the "test" for an attribute, each branch represent the result of the test, and each leaf node represents the final decision or result.
- A decision tree is constructed starting from the root node/parent node (dataset), which splits into left and right child nodes (subsets of dataset). These child nodes are further divided into their children node, and themselves become the parent node of those nodes.

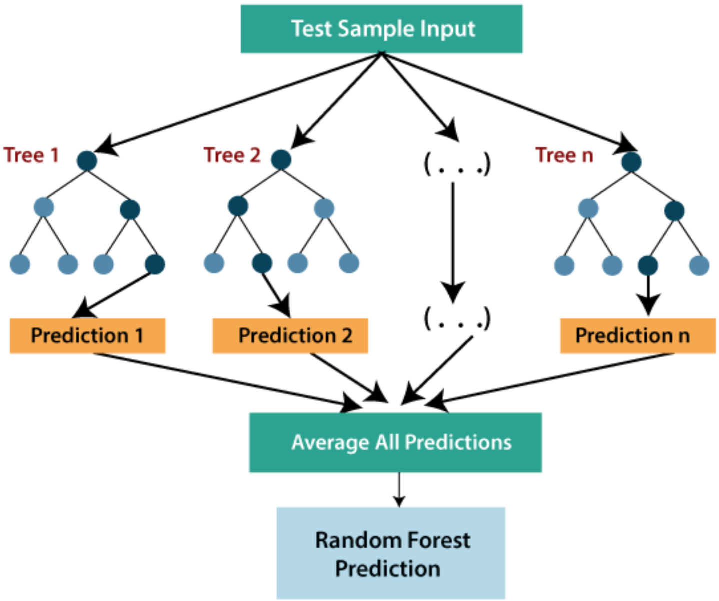


Above image showing the example of Decision Tee regression, here, the model is trying to predict the choice of a person between Sports cars or Luxury car.

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

$g(x)= f_0(x)+ f_1(x)+ f_2(x)+....$

- Random forest uses **Bagging or Bootstrap Aggregation** technique of ensemble learning in which aggregated decision tree runs in parallel and do not interact with each other.
- With the help of Random Forest regression, we can prevent Overfitting in the model by creating random subsets of the dataset.



Ridge Regression:

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

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

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

Lasso Regression:

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

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

Backward Elimination

09 July 2023 16:13

- Backward elimination is a feature selection technique while building a machine learning model.
- It is used to remove those features that do not have a significant effect on the dependent variable or prediction of output.

There are various ways to build a model in Machine Learning.

a. All-in

b. Backward Elimination

c. Forward Selection

d. Bidirectional Elimination

e. Score Comparison

Steps of Backward Elimination

Below are some main steps which are used to apply backward elimination process:

Step-1: *Firstly, We need to select a significance level to stay in the model. (SL=0.05)*
Step-2: *Fit the complete model with all possible predictors/independent variables.*
Step-3: *Choose the predictor which has the highest P-value, such that.*

1. *If P-value >SL, go to step 4.*
2. *Else Finish, and Our model is ready.*

Step-4: *Remove that predictor.*
Step-5: *Rebuild and fit the model with the remaining variables*.

Need for Backward Elimination: An optimal Multiple Linear Regression model:

- In order to optimize the performance of the model, use the Backward Elimination method.
- This process is used to optimize the performance of the MLR model as it will only include the most affecting feature and remove the least affecting feature.

Steps for Backward Elimination method:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

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

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

#Catgorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_x= LabelEncoder()
x[:, 3]= labelencoder_x.fit_transform(x[:,3])
onehotencoder= OneHotEncoder(categorical_features= [3])
x= onehotencoder.fit_transform(x).toarray()

#Avoiding the dummy variable trap:
x = x[:, 1:]
```

```


# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.2, random_state=0)

#Fitting the MLR model to the training set:
from sklearn.linear_model import LinearRegression
regressor= LinearRegression()
regressor.fit(x_train, y_train)

#Predicting the Test set result;
y_pred= regressor.predict(x_test)

#Checking the score
print('Train Score: ', regressor.score(x_train, y_train))
print('Test Score: ', regressor.score(x_test, y_test))
```

From the above code, we got training and test set result as:

Train Score: 0.9501847627493607

Test Score: 0.9347068473282446

The difference between both scores is 0.0154.

Step: 1- Preparation of Backward Elimination:

Importing the library: we need to import the statsmodels.formula.api library, which is used for the estimation of various statistical models such as OLS(Ordinary Least Square).

```
import statsmodels.api as smf
x = nm.append(arr = nm.ones((50,1)).astype(int), values=x, axis=1)
```

Step: 2:

- Now, we are actually going to apply a backward elimination process. Firstly we will create a new feature vector **x_opt**, which will only contain a set of independent features that are significantly affecting the dependent variable.
- Next, as per the Backward Elimination process, we need to choose a significant level(0.5), and then need to fit the model with all possible predictors. So for fitting the model, we will create a **regressor_OLS** object of new class **OLS** of **statsmodels** library. Then we will fit it by using the **fit()** method.
- Next we need **p-value** to compare with SL value, so for this we will use **summary()** method to get the summary table of all the values.

```
x_opt=x [:, [0,1,2,3,4,5]]
regressor_OLS=sm.OLS(endog = y, exog=x_opt).fit()
regressor_OLS.summary()
x_opt=x[:, [0,2,3,4,5]]
regressor_OLS=sm.OLS(endog = y, exog=x_opt).fit()
regressor_OLS.summary()
x_opt= x[:, [0,3,4,5]]
regressor_OLS=sm.OLS(endog = y, exog=x_opt).fit()
regressor_OLS.summary()
x_opt=x[:, [0,3,5]]
regressor_OLS=sm.OLS(endog = y, exog=x_opt).fit()
```

```
regressor_OLS.summary()
x_opt=x[:, [0,3]]
regressor_OLS=sm.OLS(endog = y, exog=x_opt).fit()
regressor_OLS.summary()
```

Estimating the performance:

Building Multiple Linear Regression model

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('50_CompList1.csv')

#Extracting Independent and dependent Variable
x_BE= data_set.iloc[:, :-1].values
y_BE= data_set.iloc[:, 1].values


# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_BE_train, x_BE_test, y_BE_train, y_BE_test= train_test_split(x_BE, y_BE, test_size= 0.2, random_state=0)

#Fitting the MLR model to the training set:
from sklearn.linear_model import LinearRegression
regressor= LinearRegression()
regressor.fit(nm.array(x_BE_train).reshape(-1,1), y_BE_train)

#Predicting the Test set result;
y_pred= regressor.predict(x_BE_test)

#Cheking the score
print('Train Score: ', regressor.score(x_BE_train, y_BE_train))
print('Test Score: ', regressor.score(x_BE_test, y_BE_test))
```

Output:

After executing the above code, we will get the Training and test scores as:

Train Score: 0.9449589778363044

Test Score: 0.9464587607787219

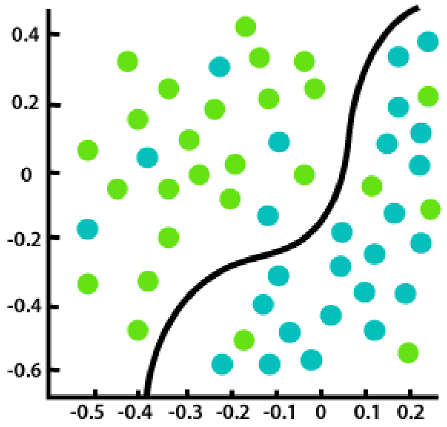
the training score is 94% accurate, and the test score is also 94% accurate. The difference between both scores is .00149. This score is very much close to the previous score, i.e., 0.0154, where we have included all the variables.

We got this result by using one independent variable (R&D spend) only instead of four variables. Hence, now, our model is simple and accurate.

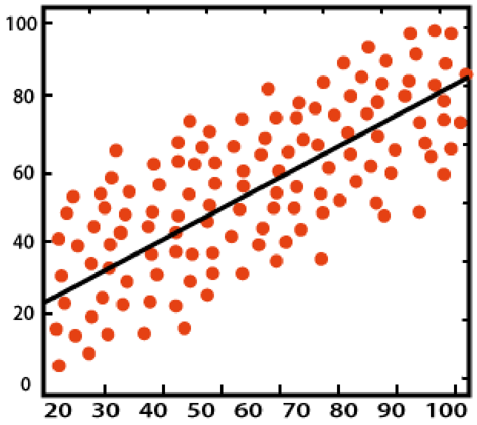
Regression vs. Classification

16 July 2023 19:03

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



Classification



Regression

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

Some Questions

14 September 2023 12:15

1. **What is a "model" in machine learning?**
   A model is a mathematical relationship derived from data that an ML system uses to make predictions

2. **If you wanted to use an ML model to predict energy usage for commercial buildings, what type of model [Regression/classification] would you use?**
   Regression - *Energy usage is measured in kilowatt-hours (kWh), which is a number, so you'd want to use a regression model.*

3. **What distinguishes a supervised approach from an unsupervised approach?**
   A supervised approach is given data that contains the correct answer. The model's job is to find connections in the data that produce the correct answer. An unsupervised approach is given data without the correct answer. Its job is to find groupings in the data.

4. **What attributes of a dataset would be ideal to use for ML?**
   Large size / High diversity
   A large number of examples that cover a variety of use cases is essential for a machine learning system to understand the underlying patterns in the data. A model trained on this type of dataset is more likely to make good predictions on new data.

5. **Why does a model need to be trained before it can make predictions?**
   A model needs to be trained to learn the mathematical relationship between the features and the label in a dataset.