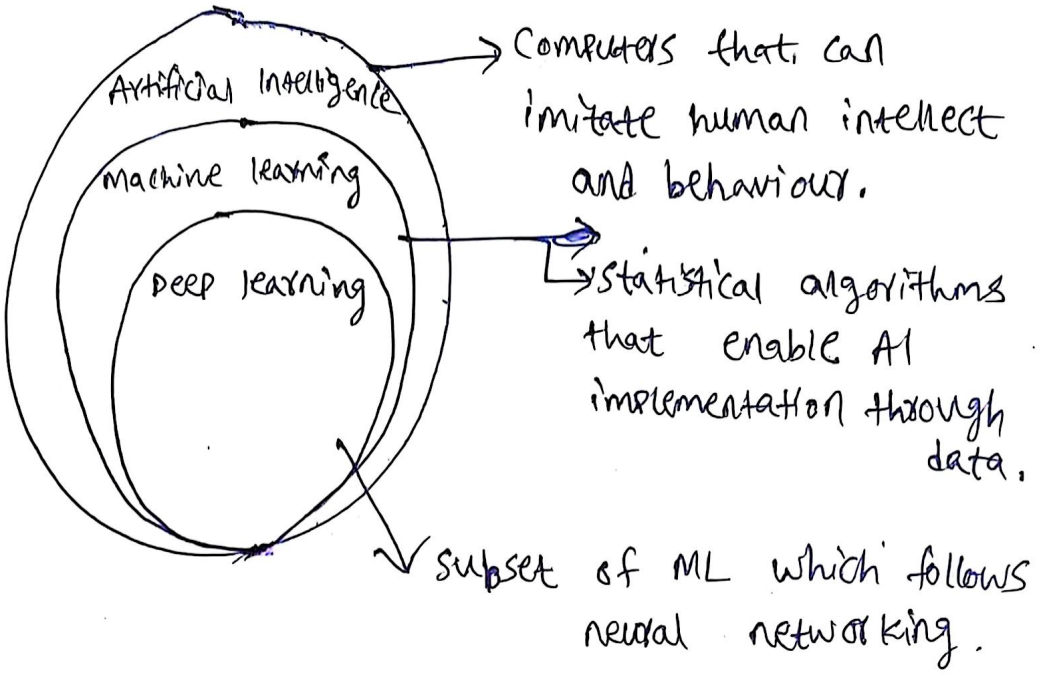
AI vs ML vs DL



AI (Artificial intelligence)

→ AI is basically the mechanism to incorporate human intelligence into machines through a set of rules (algorithms).

→ AI is the broader family consisting of ML and DL as its components.

→ The efficiency of AI is basically the efficiency provided by ML and DL respectively.

→ Examples – Amazon's Alexa, Siri

ML (Machine learning)

→ ML is the study that uses statistical methods enabling machines to improve with experience.

→ ML is subset of AI.

→ Less efficient than DL as it can't work for longer dimensions or higher amount of data.

→ Examples → Car price prediction, Diabetes prediction.

DL (Deep learning)

→ DL makes use of neural networks (similar to human neurons) to imitate functionality just like a human brain.

→ DL is the subset of ML.

→ More powerful than ML as it can easily work for larger set of data.

→ It attains the highest rank in terms of accuracy when it is trained with large amount of data.

→ Examples - Image classification, Image de-noising etc.

Note!

** DALL-E and DALL-E 2 are machine learning models developed by openAI to generate digital images from natural language descriptions.

**

Variables = features = Dimensions = Attributes = Column

Records = rows

Supervised Learning:

→ It is the learning of the model where with input variable (say, X) and an output variable (say, Y) and an algorithm to map the input to the output i.e. $Y = f(X)$.

→ The basic aim is to approximate the mapping function (mentioned above) that when there is new input data (x) then the corresponding o/p variable can be predicted.

→ It is called supervised learning because the process of learning (from the training dataset) can be thought of as a supervisor who is supervising the entire learning process. Thus, the "learning algorithm" iteratively makes predictions on the training data and is corrected by the "supervisor", and the learning stops when the algorithm achieves an acceptable level of performance

→ It is called supervised because data is labelled (y column is present)

→ Examples of supervised learning algorithms – Regression and Classification

$$Y = f(X)$$

Y = dependent variable

X = independent variable

*** If there are two categories in target variable, then it is binary classification.

Terminologies:

1) Training Data – The Data onto which, we will train our ML model

2) Testing Data – The Data onto which the ML model (already trained on training data) will be used to generate predictions.

Prediction Pipeline:

1) Load the data

2) Feature Engineering – data cleansing and domain understanding
(a) Removal / Filling of null values

b) Removal of duplicates (rows or columns)

c) Encoding categorical variables.

d) Standardizing or scaling variables.

Age (10-99)

Income — 100, 1000, 10000, - - - -

e) Creating custom features out of existing variables

Date (2019-03-28) ⇒ from Date, we can create custom features Year, Quarter, Month, Day

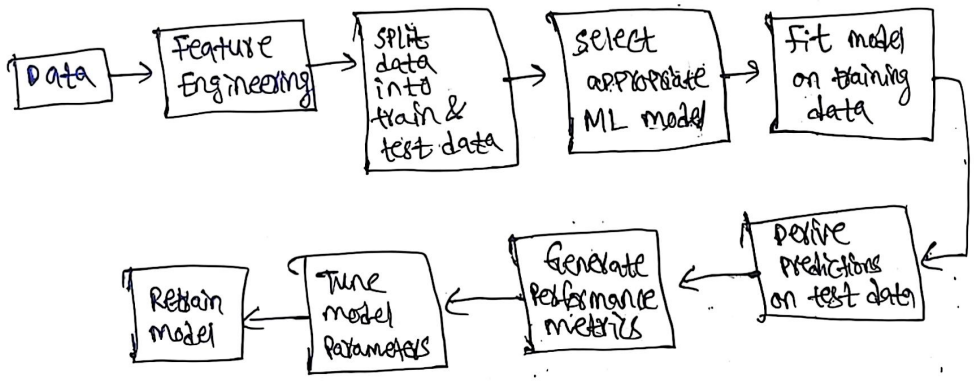
f) Binning

Salary — 12K, 15K, 20K, 18K, 25K, - - - -

Salary_bracket —

Low (10K-24K) med (25K-49K), High (50K-70K)

Prediction pipeline!



Unsupervised Learning!

→ Unsupervised learning is where only the input data (say, X) is present and no corresponding output variable is present.

→ The main aim of Unsupervised learning is to model the distribution in the data in order to learn more about the data.

⇒ It is called so, because there is no correct answer and there is no such supervisor (unlike supervised learning algorithms

are left to their own devises to the interesting structure in the

→ It is unsupervised as data

→ Examples— Clustering and Di

Clustering Example!

X (X1, X2)

After clustering

X1

Y1

* After performing becomes supe

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

DataFrame:

→ Data represented in rows o

→ DataFrame consists of inde

→ A " can " of a

Series:

→ Single column data is

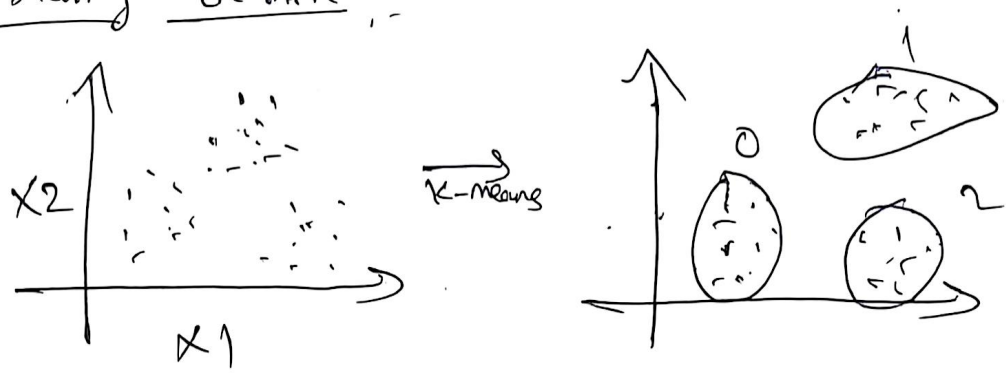
→ Series also bear index

are left to their own devises to discover and present the interesting structure in the data.

→ It is unsupervised as data is not labelled. (Y column is present).

→ Examples- Clustering and Dimensionality Reduction.

Clustering Example!-



X (X1, X2)

After clustering

X : [X1, X2]

Y : (0, 1, 2)

* After performing clustering the data becomes supervised for classification

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

DataFrame:

→ Data represented in rows and columns.

→ DataFrame consists of index as well.

→ A " can " of a single column.

Series:

→ single column data is series

→ series also bear index as well.

P)
```
data = {'Name' : ['Ankit', 'Shubham', 'Mohit', 'Shreya'],
        'Age' : [23, 26, 34, 20]
        'City' : ['Delhi', 'Bangalore', 'Delhi', 'Pune']}
df = pd.DataFrame(data)
df
```

O/P :-

	Name	Age	City
0	Ankit	23	Delhi
1	Shubham	26	Bangalore
2	Mohit	34	Delhi
3	Shreya	20	Pune

P) df.columns

O/P :- Index(['Name', 'Age', 'City'], dtype='object')

P) df.values

O/P :-
```
([['Ankit', 23, 'Delhi'],
  ['Shubham', 26, 'Bangalore'],
  ['Mohit', 34, 'Delhi'],
  ['Shreya', 20, 'Pune']], dtype=object)
```

P) df.shape

O/P :- (4, 3)

P) df.sample() → if we pass any number here, that number of random rows will displayed.

O/P :- It returns any random row

P) print(type(df))

O/P:-

<class 'pandas.core.frame.DataFrame'>

P) Accessing single column

df['Name']

O/P:

```
0    Ankit
1    Shubham
2    Mohit
3    Shreya
Name: Name, dtype: object
```

P) Print type(df['Name'])

O/P: <class 'pandas.core.series.Series'>

#Selecting multiple columns

P) df[['Name', 'City']]

O/P:

	Name	City
0	Ankit	Delhi
1	!	!
2	(	(
3	Shreya	Pune

Regression:

→ It is a predictive modeling technique which investigates the relationship b/w dependent & independent variable (one or more)

→ dependent variable is continuous in nature. eg - sales, weight, Profit, Revenue, Price etc...

→ y = dependent variable / output
x = independent variable / input

Simple Linear Regression:-

→ It is a regression model that estimates the relationship between one independent variable and one dependent variable using a straight line.

→ $y = ax+b$ or $y = mx+c$

where

x = independent variable / input feature / input attribute / column

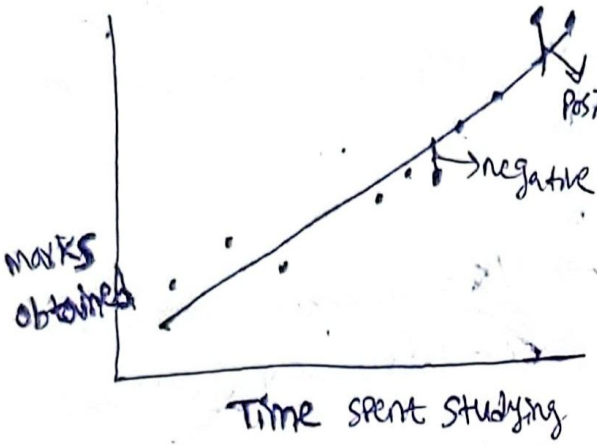
y = dependent variable / output feature / o/p attribute / column

a/m = slope or coefficient or weight or how much we expect y to change as x changes.

b/c = intercept / constant / bias

$$m = a = \frac{(y_2 - y_1)}{(x_2 - x_1)} = \frac{dy}{dx}$$

Line of best fit :-

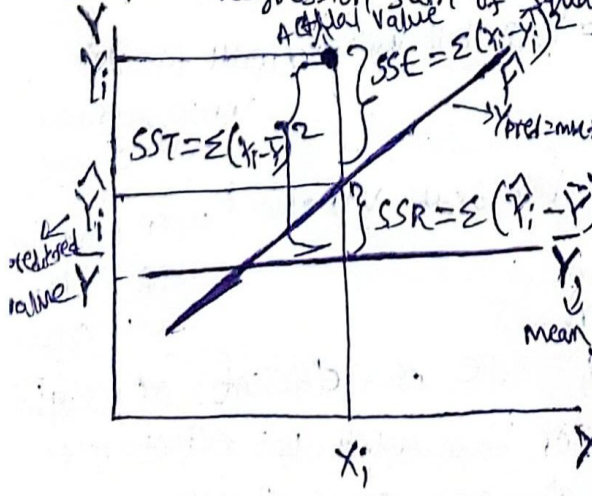


* The linear Regression model finds the best fit line that minimizes the squared error b/w the actual data points and its $\hat{y}$ dropped on the predicted line.

SST = Total sum of squares

SSE = Error / Residual sum of squares

SSR = Regression sum of squares.



Total variation is made up of two parts!

$$SST = SSE + SSR$$

$$SST = \Sigma (Y_i - \bar{Y})^2$$

$$SSR = \Sigma (\hat{Y}_i - \bar{Y})^2$$

$$SSE = \Sigma (Y_i - \hat{Y}_i)^2$$

Where

$\bar{Y}$ = Average value of dependent variable

Y_i = Observed values of dependent variables

$\hat{Y}_i$ = Predicted value of Y for the given X_i value.

R2 score (Coefficient of determination or Goodness of fit) :-

→ The coefficient of determination is the portion of the total variation in the dependent variable that is explained by variation in the independent variable.

→ This is used to evaluate the performance of the model

→ The coefficient of determination is also called r-squared and is denoted as r^2.

$$r^2 = \frac{SSR}{SST} = \frac{\text{regression sum of squares}}{\text{total sum of squares}}$$

note : $0 \leq r^2 \leq 1$

→ $SST = SSE + SSR$

$$R^2 \text{score} = SSR/SST = \frac{(SST - SSE)}{SST} = 1 - \frac{SSE}{SST}$$

When SSE = 0, R^2 score = 1 (Best case scenario)

When SSE = SST, R^2 score = 0 (worst case scenario)

Regression Metrics:-

Mean Absolute error (MAE)

It is the mean of the absolute value of the errors.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Mean Squared Error (MSE)

It is the mean of the squared errors.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{SSE}{n}$$

Root Mean Squared Error (RMSE)

It is the square root of the squared errors.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} = \sqrt{MSE}$$

$$= \sqrt{\frac{SSE}{n}}$$

Multiple Linear Regression:-

→ MLR is used to estimate relationship b/w two or more independent variables and one dependent variable.

→ It has an eqn of the form

$$y = m_1x_1 + m_2x_2 + \cdots + m_nx_n + b$$

where

$x_1, x_2 \cdots, x_n$ = independent variables / input features.

y = dependent variable / output feature

$m_1, m_2 \cdots, m_n$ = coefficients / slope corresponding to $x_1, x_2 \cdots, x_n$

b = intercept / constant / bias

→ It is practically out of scope to represent multiple linear regression on a scatter chart with actual data points and a regression line as the scatter chart would have to span multiple dimensions corresponding to each independent variable and likewise the Regression line would also span across multiple dimensions.

↓

This paragraph means that if we have 5 dimensions, we can't plot them in 2 dimensional graph.

Pros:-

→ Linear Regression Performs well when the dataset is linearly dependent.

→ Linear Regression is easy to implement, interpret and

efficient to train.

Cons :-

→ main limitation of Linear Regression is the assumption of linearity between the dependent variable and the independent variables. In the real world, the data may not always be linearly dependent.

For every regression model :

→ model only accepts int or float as input for x and y

→ Model will not accept any null value.

→ X has to be either a dataframe or a 2D numpy array or a list of list.

→ Y has to be either a series or a 1D numpy array or a list.

Dataframe :-

(a) In this, data is represented in rows and columns with index. Generally Dataframe consists of multiple column data.

b) Dataframe can be of single column with index, however

Series

→ Single column data with index.

→ Series cannot contain multiple column, however it can contain multiple indexes.

Simple Linear Regression Implementation :-

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

1) Load the data

```
d = {'Area(sqft)' : [2600, 3000, 3200, 3600, 4000],
     'price' : [550000, 565000, 610000, 680000, 725000]}
```

```
print(d)
print(type(d))
```

O/P :-

```
{'Area(sqft)' : [---], 'price' : [---]}
<class 'dict'>
```

2) `df = pd.DataFrame(d)`

`df`

O/P :-

	Area(sqft)	price
0	2600	
1		
2		
3		
4		

Prbm statement – Based on Area(sqft) predict the price when

a) Area = 4400 sqft

b) Area = 3450 sqft.

P) df['price']

O/P)
```
0    550000
1
2
3
4
Name: price, dtype: int64
```

P) single column dataframe.

```
df[['price']]
type(df[['price']])
```

O/P)

	price
0	550000
1	
2	
3	
4	

P) df.shape

O/P) (5,2)

P)
```
X = df[['Area(sqft)']]
Y = df['price']
print(type(x))
print(type(y))
```

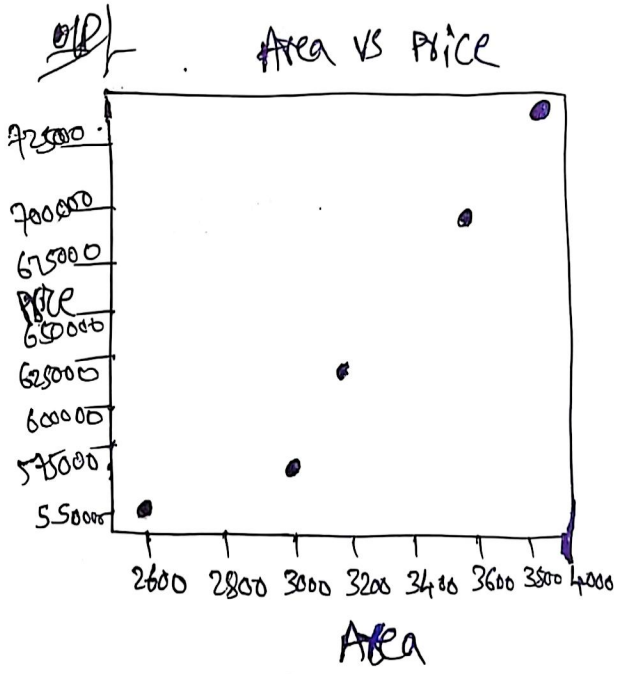
O/P)
```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
```

P) print(x.shape)
print(y.shape)

O/P)
```
(5,1)
(5,)
```

P)
```
plt.scatter(df['Area(sqft)'],
            df['price'], color='maroon')
plt.xlabel('Area')
plt.ylabel('price')
plt.title('Area vs Price')
plt.show()
```

O/P)



Lets create linear regression model.

P)
```
from sklearn.linear_model import
                    LinearRegression
m1 = LinearRegression()
m1.fit(X,Y)
```

#model.fit() – It trains the ML model on the training data.

#model.score() gives R^2 value for Regression.

P) `print ('Model score', m1.score(X,Y))`

O/P :-

Model score 0.9584301181994686

P) `Ypred_m1 = m1.predict(X)`

`Ypred_m1`

O/P :-

array([533664.38356164, 587979.45205479, ..., ..., ...])

Adding Ypred_m1 column to dataframe.

P) `df['Pred_Price'] = Ypred_m1`

`df.head()`

O/P :-

	Area(sqft)	Price	Pred_Price
0	2600	550000	533664.38356
1			
2			
3			
4			

P) `plt.scatter(df['Area(sqft)'], df['Price'], color='maroon', label='Actual data points')`

`plt.plot(df['Area(sqft)'], df['Pred_Price'], color='blue', label='Predicted line')`

`plt.xlabel('Area')`

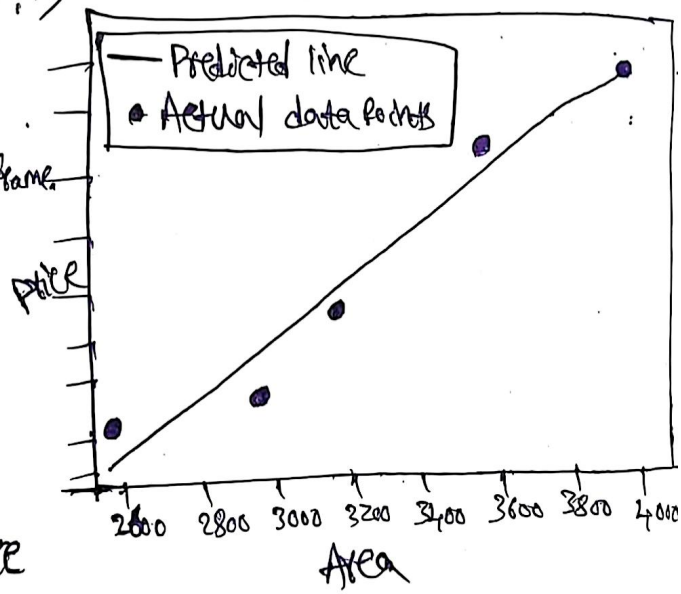
`plt.ylabel('Price')`

`plt.title('Area vs Price')`

`plt.legend()`

`plt.show()`

O/P :-



P) `# Ypred = mx+c`

`m = m1.coef_`

`c = m1.intercept_`

`print ('Slope', m)`

`print ('Intercept', c)`

O/P :-

Slope [135.78767123]

Intercept 180616.43835616432

p) df['pred_price_eqn'] = df['Area(sqft)']*m+C.
df.head()

o/p:

	Area(sqft)	Price	Pred_Price	Pred_Price_eqn
0	2600	550000	533664.383562	533664.383562
1	3000		587979.452055	587979.452055
2	3200			
3				
4			723767.123288	723767.123288

p) from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

```
def eval_metrics(ytest, ypred):
    mae = mean_absolute_error(ytest, ypred)
    mse = mean_squared_error(ytest, ypred)
    rmse = np.sqrt(mean_squared_error(ytest, ypred))
    r2s = r2_score(ytest, ypred)
    print('MAE', mae)
    print('MSE', mse)
    print('RMSE', rmse)
    print('R2_score', r2s)
```

p) eval_metric(y, ypred_m1)

MAE 11246.57534246575
MSE 186815068.4931509
RMSE 13668.030893042014
R2_score 0.9584301138199486

p) # a) Area = 4400 sqft.
b) Area = 3450 sqft.

ypred_4400_a1 = m1.predict([[4400]])
ypred_4400_a2 = 4400*m+C
print(ypred_4400_a1)
print(ypred_4400_a2)

o/p:

[778082.19178082]
[778082.19178082]

Multiple linear Regression

```
df1 = pd.read_csv('houseprice2.csv')
df1.head()
```

O/P :-

	area	bedrooms	age	price
0	2600	3.0	20	550000
1	3000	4.0	15	565000
2	3200	NaN	18	
3	3600	3.0	30	
4	4000	5.0	8	

To check if null values is present or not.

```
df1.isnull().sum()
```

O/P :-

```
area        0
bedrooms    1
age         0
price       0
dtype: int64
```

```
print(df1['bedrooms'].mean())
print(df1['bedrooms'].median())
```

O/P :-

```
4.2
4.0
```

Filling the null values.

```
df1['bedrooms'].fillna(df1['bedrooms'].median(), inplace=True)
df1.isnull().sum()
```

O/P

```
area        0
bedrooms    0
age         0
price       0
```

Confusion Matrix :-

1) It is a performance metric for classification statements.

2) It is a square matrix made up of 4 terms - TP, TN, FP, FN.

3) If target variable has n categories then shape of the confusion matrix will be (n, n)

Terminologies

1) TP (True Positive) -

Actual value is positive, ML model also predicted a positive value.

2) FN (False negative) -

Actual value is positive, ML model predicted a negative value

3) FP (False Positive) -

Actual value is negative, ML model predicted a positive value

4) TN (True Negative)

Actual value is negative, ML model also predicted a negative value.

Note!

→ Sum of all the actual positive values/cases = TP + FN

→ Sum of all the actual negative values/cases = FP + TN

→ Sum of all the positively predicted values/cases = TP + FP

→ Sum of all the negatively predicted values/cases = FN + TN

```
#x=['Age','Gender','BMI','Body_wgt',
    'Blood-Glucose_Level']
#y = Diabetic (0) or Non-diabetic (1)

#Binary Classification Example
#0 - Positive(True), 1-negative(False).
y_true=[1,0,1,1,0,0,1,0,0,1,0,1,1,0,1,0,
        0,1]
#actual vals
y_pred=[0,1,0,1,0,0,1,1,0,1,1,0,0,1,0,
        1,0,0]
#predicted values
print(len(y_true),len(y_pred))
```

O/P:-

18 18

Classification metrics :-

1) Precision - TP/(TP+FP), TN/(TN+FN)

→ From all the positively predicted cases, how many are actually positive

→ From all the negatively predicted cases, how many are actually negative

2) Recall - TP/(TP+FN), TN/(TN+FP)

→ From all the actual positive cases, how many has the ML model predicted positive.

→ From all the actual negative cases, how many has the ML model predicted negative.

3) F1-score - $\frac{2 * \text{precision} * \text{Recall}}{(\text{Precision} + \text{Recall})}$

It is the harmonic mean between Precision and Recall.

4) Accuracy - $\frac{(TP+TN)}{(TP+FN+FP+TN)}$

→ Out of all the values, how many the model has predicted correctly.

```
print(y_true)
print(y_pred).
```

O/P:-

[1, 0, 1, 1, 0, 0, ---, 0, 1]

[0, 1, 0, 1, 0, 0, ---, 0, 0]

P) from sklearn.metrics import confusion_matrix, classification_report

cm = confusion_matrix(y_true, y_pred)

AV = 0 , PV = 0 ⇒ TP = 4

AV = 0 , PV = 1 ⇒ FN = 5

AV = 1 , PV = 0 ⇒ FP = 6

AV = 1 , PV = 1 ⇒ TN = 3

print(cm)

[TP FN]

[FP TN]

O/P:- [[4 5]
[6 3]]

P) print(classification_report(y_true, y_pred))

	precision	recall	f1-score	support
(Diabetic) 0	0.40	0.44	0.42	9
(Non-Diabetic) 1	0.38	0.33	0.35	9
accuracy			0.39	18
macro avg	0.39	0.39	0.39	18
weighted avg	0.39	0.39	0.39	18

precision - $\frac{TP}{(TP+FP)}$, $\frac{TN}{(TN+FN)}$

Recall - $\frac{TP}{(TP+FN)}$, $\frac{TN}{(TN+FP)}$

f1-score - $\frac{2 * \text{precision} * \text{Recall}}{(\text{precision}+\text{Recall})}$

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

Validation :-

```
print(cm)
```

op) `[[4 5]` # [TP FN]
`[6 3]]` # [FP TN]

$prec0 = \frac{4}{(4+6)}$

$prec1 = 3/(3+5)$

$rec0 = 4/(4+5)$

$rec1 = 3/(3+6)$

Print values

op:

prec0 0.4

prec1 0.375

Rec0 0.4444--

Rec1 0.3333--

p) $fs0 = 2 * prec0 * rec0 / (prec0 + rec0)$

$fs1 = 2 * prec1 * rec1 / (prec1 + rec1)$

Print

op)

f1_score0 0.42105263

f1_score1 0.35294

p) $acc = (4+3)/(4+3+5+6)$

print(acc)

op:

0.3888

Logistic Regression

→ It is a supervised learning algorithm used for classification.

→ LR is a classification algorithm used to assign observations to a discrete set of classes.

→ It is used when the dependent variable (target) is categorical.

→ LR transforms its o/p using the logistic or sigmoid function to return a probability value which can then be mapped to two or more discrete classes.

→ It is approximate algorithm when dependent variable is categorical and consists of two categories (binary).

→ It is a special case of linear regression where the target variable is categorical in nature. It uses a log of odds as the dependent variable.

→ Logistic Regression predicts the probability of occurrence of a binary event utilizing a logit or sigmoid function.

→ Odds Ratio = $\frac{P(\text{event happening})}{1-P(\text{event happening})}$

→ log of odds = $\log\left(\frac{p}{1-p}\right) = Y = a+bX$

Derivation of sigmoid function:

$$\log\left(\frac{p}{1-p}\right) = ax+b$$

$$p/(1-p) = e^{ax+b}$$

$$p = e^{ax+b} - p\,e^{ax+b}$$

$$p\left(1+e^{ax+b}\right) = e^{ax+b}$$

$$p = \frac{e^{ax+b}}{1+e^{ax+b}}$$

$$p = \frac{1}{1+e^{-ax+b}}$$

↓

This eqn is called logistic or sigmoid or Logit function

$$S(z) = \frac{1}{1+e^{-z}}$$

where $z = ax+b$

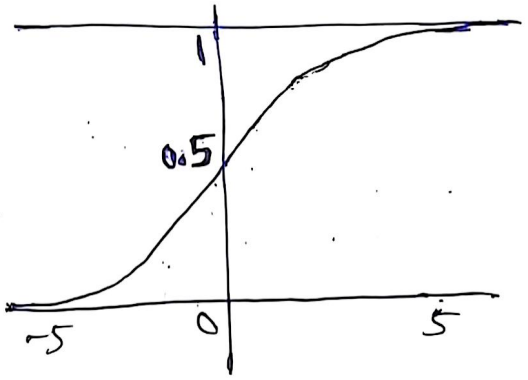
Sigmoid function:-

→ In order to map predicted values to probabilities, we use the sigmoid function. The function maps any real value into another value between 0 and 1. In machine learning, we use sigmoid to map predictions to probabilities.

→ The sigmoid function, also called logistic function gives an 'S' shaped curve that can take any real-valued number and map it into a value blw 0 & 1.

→ If the curve goes to positive infinity, y predicted will become 1, and if the curve goes to negative infinity, y predicted will become 0.

→ If the o/p of sigmoid function is more than 0.5, we can classify the outcome as 1 or Yes and if it is less than 0.5, we can classify it as 0 or NO.



Pros/Advantages

→ Logistic regression is easier to implement, interpret and very efficient to train.

→ It provides good accuracy for many simple data sets and it performs well when the dataset is linearly seperable.

→ Due to its simple probabilistic interpretation, the training time of logistic regression algorithm comes out to be far less than most complex algorithms

Cons/Disadvantages

→ only important and relevant features should be used to build the model otherwise the probabilistic predictions made by the model may be incorrect and the model's predictive value may degrade.

→ Doesn't perform well with non-linearly seperable data

linearly seperable

Non-linearly seperable

for every classification model:

→ Model only accepts int or float as input for X.

→ Model can accept input as strings/object (in certain models) as input for y

→ Model will not accept any null value.

→ X has to be either a Dataframe or a 2D numpy array of a list of list.

→ y has to be either a series or a 1D numpy array or a list.

import libraries

P)
```
df = pd.read_excel('insurance_data.xlsx')
df.head()
```

O/P

	age	bought_insurance
0	22	0
1	25	0
⋮	⋮	⋮
4	46	1

P) df.shape

O/P

(27,2)

P) df.isnull().sum()

O/P

age 0
bought_insurance 0
dtype: int64

P) df.dtypes

age int64
bough_insu int64
dtype: object

int or

P)
```
X = df[['age']]
y = df['bough_insu']
print(X.shape)
print(y.shape)
print(type(X))
print(type(y))
```

O/P

(27,1)
(27,1)
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>

P) 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)
Print(X_train.shape)
      (X_test.shape)
      (Y_train.shape)
      (Y_test.shape)
```

O/P)

(20,1)
(7,1)
(20,)
(7,)

P) X_train.head()

O/P)

	age
20	21
9	61
5	51
11	28
8	62

→ Every output changes for above program because training is based on different rows

P) from sklearn.linear_model import LogisticRegression.

```
ml = LogisticRegression()
ml.fit(X_train, Y_train)
#accuracy values
print('Training score', ml.score(X_train, Y_train))
print('Testing score', ml.score(X_test, Y_test))
```

O/P) Training score 0.9

P) ypred_m1 = m1.predict(x_test)

ypred_m1

O/P:

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

P) from sklearn.metrics import confusion_matrix, classification_report.

cm = confusion_matrix(y_test, ypred_m1)

print(cm)

print(classification_report(y_test, ypred_m1))

O/P:

```
[[2 1]
 [0 4]]
```

	precision	recall	f1-score	support
0	1.00	0.67	0.80	3
1	0.80	1.00	0.89	4
accuracy			0.86	7
macro avg	0.90	0.83	0.84	7
weighted avg	0.89	0.86	0.85	7

P) m = m1.coef_

c = m1.intercept_

print(m)

print(c)

O/P:

```
[[0.1575114]]
[-5.81788525]
```

P) # sigmoid = 1/(1+e^(mx+c))

```
def sigmoid(x, m, c):
    logistic = 1/(1+np.exp(-(m*x+c)))
    print(logistic)
```

Problem Statement :

- Generate predictions whether or not the person bought the insurance for given age.

(a) Age = 59

(b) Age = 23

P) ypred_59 = mn.predict([[59]])
print(ypred_59)
sigmoid(59,m,c)

O/P :
[1]
[[0.96997638]]

P) ypred_23 = mn.predict([[23]])
print(ypred_23)
sigmoid(23,m,c)

O/P :
[0]
[[0.100189 3]]

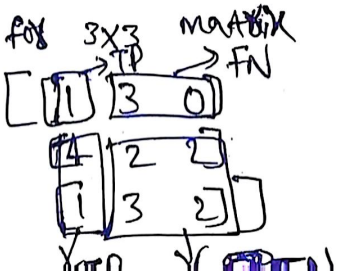
Multiclass classification :

y_true = [1,0,1,2,0,2,1,0,2,1,0,1,1,2,1,2,2,1]
y_Pred = [0,1,2,1,0,2,1,1,0,1,1,0,2,1,0,1,2,0]
print(len(y_true), len(y_Pred))

O/P : 18 18

P) cm2 = confusion_matrix(y_true, y_Pred)
print(cm2)

O/P : [[1 3 0]
[4 2 2]
[1 3 2]]



print(classification_report(y_true, y_pred))

	precision	recall	f1-score	support
0	0.17	0.25	0.20	4
1	0.25	0.25	0.25	8
2	0.50	0.33	0.40	6
accuracy			0.28	18
macro avg	0.31	0.28	0.28	18
weighted avg	0.31	0.28	0.29	18

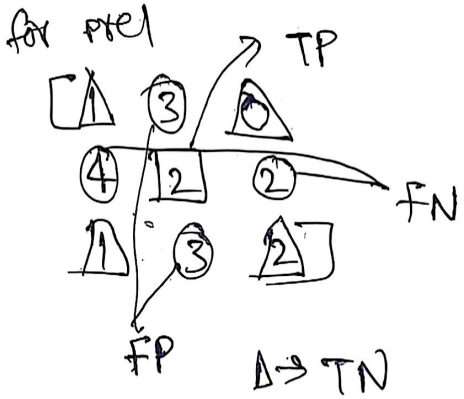
Precision — TP/(TP+FP) , TN/(TN+FN)

pre0 = 1/(1+5)

pre1 = 2/(2+6)

pre2 = 2/(2+2)

print pre0
pre1
pre2



OR

0.1666
0.25
0.5

Decision Trees

→ It falls under the category of supervised ML.

→ It is used for both classification and Regression.

→ A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes).

→ The decision tree from the name itself signifies that it is used for making decisions from the given dataset. The concept behind the decision tree is that it helps to select appropriate features for splitting the tree into ~~different parts~~ subparts.

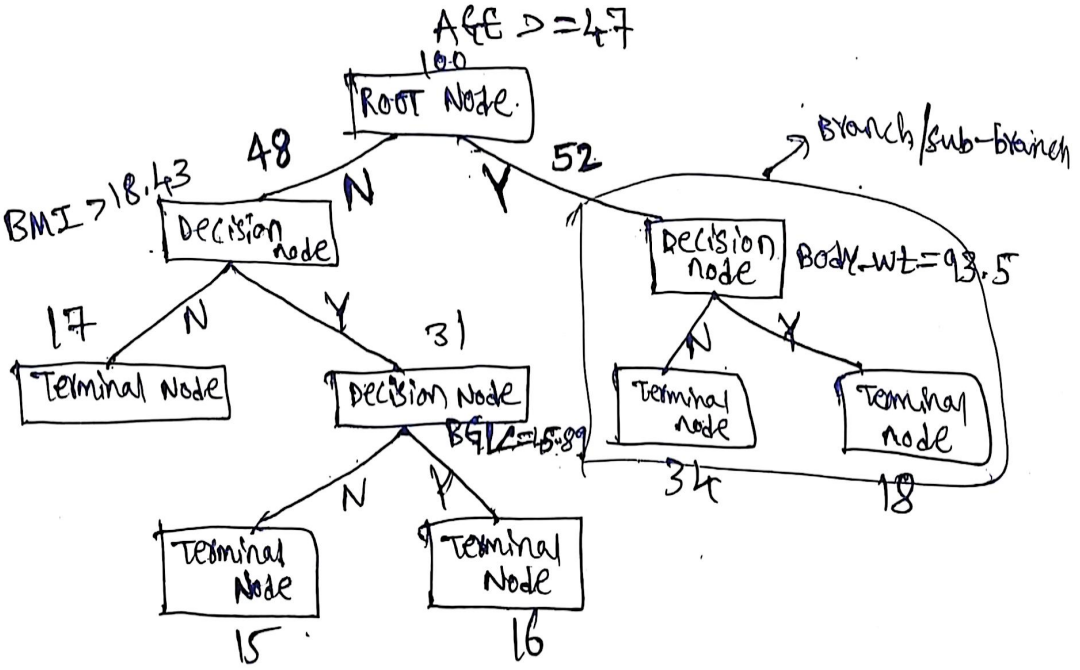
→ It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes.

→ Decision tree uses the tree representation to solve the problem in which each leaf node corresponds to a class label (target variable) and attributes are represented on the internal node of the tree.

Example:-
We have 100 rows of data with

X = 'Age', 'Gender', 'BMI', 'Body_wt', 'Blood_Glucose_Level'.

Y = 0 or 1 (0 = Non diabetic, 1 = Diabetic)



Decision Tree Algorithm:

→ We use statistical methods (gini or entropy) for ordering attributes as root or the internal node.

E(Target_variable) = $\Sigma -p \log p$

E(Target_variable, feature) = P(feature) * E(Target_variable, feature)

Information Gain = Entropy(Target_variable) − Entropy(Target_variable, feature).

1) Entropy:

→ It is the measure of uncertainty or disorder or impurity. It characterizes the impurity of an arbitrary collection of examples. The higher the entropy more the information content.

→ If the sample is completely homogenous the entropy is zero and if the sample is equally divided it has entropy of one.

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

where

p_i = probability of label i

n = number of categories in the label

Information Gain:

→ It is based on decrease in entropy after dataset is split on an attribute/feature or column)

→ The split with highest information

2) Gini

→ Gini index is a metric to measure how often a randomly chosen element would be incorrectly identified.

→ It means an attribute with lower Gini index should be preferred.

→ $Gini = 1 - \Sigma (p_i)^2$ for i=1 to no. of classes

Pros:

→ Works well with numerical & categorical features.

→ Decision Tree can handle non-linear data.

→ Performs well on large datasets.

Cons:

→ It suffers from overfitting. Random Forest mitigates this issue

→ Training time is high.

Overfitting: → scattering b/w actual value & predicted value

how far is the actual value from predicted value

a) High variance and low bias

b) High training score but low test score.

Parameters :-

1) Criterion - entropy or gini

2) max_depth - max depth of decision tree.

3) min_samples_split - Minimum number of samples beyond which an internal node gets converted to a leaf node.

Random Forest Classification

1) RF is an ensemble machine learning technique capable of performing both regression and classification tasks using multiple decision trees and a statistical technique called Bagging (Bootstrap Aggregation).

2) A RF besides just averaging the prediction of trees it uses two key concepts that give it the name random:
   (a) Random sampling of training observations (rows) when building trees.
   (b) Random subsets of features (columns) for splitting nodes.

3) Random forest builds multiple decision trees and merge their predictions together to get a more accurate and stable prediction rather than relying on individual decision trees.

4) It combines a number of weak and merges them into a strong estimator.

5) RF, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction.

6) Random forest intuition - A large no. of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models.

7) It uses Bootstrapping (selection with replacement) to select random subsets of records and/or columns for creating individual decision trees. It means some samples might be used multiple times in a single tree.

Example :-

1) There are 100 rows of data.

2) The target variable (Y) has 2 categories - A and B

3) In random forest, we are constructing 7 DTs.

4) Suppose out of 7 DTs, row 37 is a part of DT1, DT3, DT4, DT5, DT7.

5) predictions generated by decision trees for row no 37 are:

DT1 → A
DT3 → B
DT4 → A
DT5 → A
DT6 → B

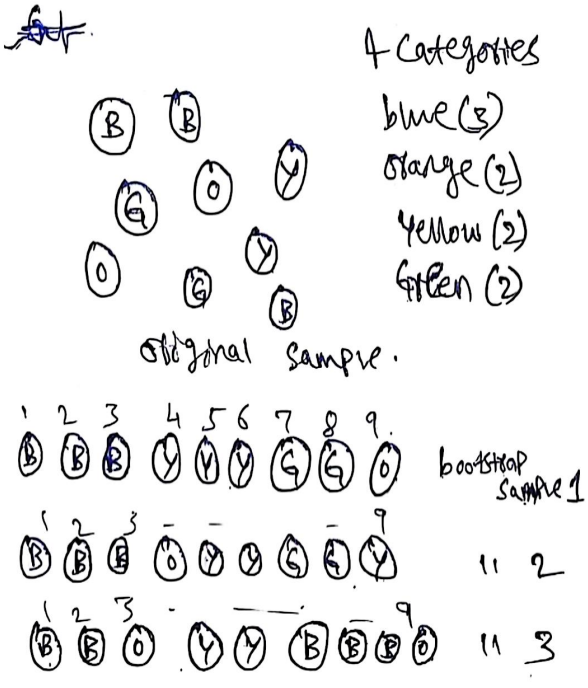
6) Prediction out of RF for row no 37 is mode of these predictions (3A and 2B)

→ mode (3, 2) = 3.

7) Prediction for Row no 37 is Category A according to RF;

Bootstrapping :-

→ It is randomly selecting samples from training data with replacement. The samples so generated are called bootstrap samples.



Working of Random forest :-

1) Select random samples from a given dataset.

2) Construct a decision tree for each sample and get a prediction result from each decision tree.

3) Perform a vote for each predicted result.

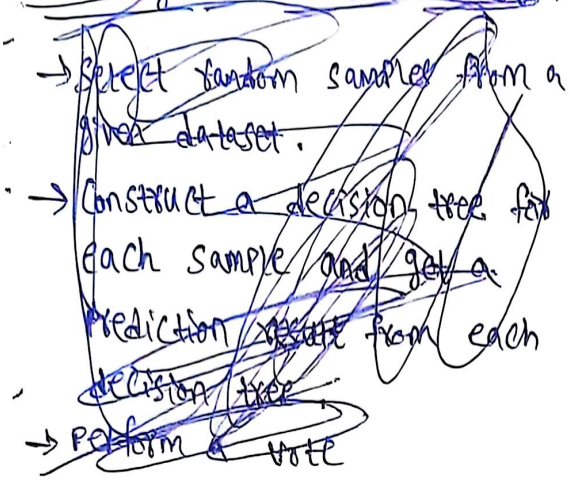
4) Select the prediction result with the most votes as the final prediction.

(a) The random forest combines hundreds or thousands of decision trees, trains each one on a slightly different set of observations, splitting nodes in each tree considering a limited number of features.

(b) The final predictions of the random forest are made by averaging the predictions of each individual tree.

(c) The random forest is a classification algorithm consisting of many decision trees. It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree.

Working of Random forest :



→ Select random samples from a given dataset.
→ Construct a decision tree for each sample and get a prediction result from each decision tree.
→ Perform a vote

Parameters of Rf :-

1) n_estimators : This is the no. of trees you want to build before taking the maximum voting or averages of predictions. Higher no. of trees give you better performance but makes your code slower.

2) min_samples_split : The minimum no. of samples required to split an internal node. The algorithm keeps splitting the nodes as long as a node has more samples (data points) than the number specified with min_samples_split Parameter.

3) Criterion : gini or entropy.

4) max_features : These are the maximum no. of features Rf is allowed to try in individual tree. The values it can have are :-

a) Auto/None : This will simply take all the features which make sense in every tree. Here we simply do not put any restrictions on the individual tree.

b) sqrt : This option will take square root of total no. of features in individual run. For instance, if the total no. of variables are 100, we can only take 10 of them in individual tree. "log2" is another similar type of option for max_features.

c) 0.x : This option allows the random forest to take x% of variables in individual run. We can assign and value in a format "0.x" where we want x% of features to be considered.

5) bootstrap : true/false.

If it is set to true, samples are drawn with replacement and samples are drawn without replacement if bootstrap is set to false.

Pros :-

→ Rf is considered as a highly accurate and robust method bcz of no. of decision trees participating in the process.
→ It overcomes the problem of overfitting faced by decision tree. The main reason is that it takes the average of all the predictions.

Cons:-

→ RF is slow in generating predictions bcz. it has multiple decision trees. Whenever it makes a prediction, all the trees in the forest have to make a prediction for the same given input and then perform voting on it. This whole process is time-consuming.

Implementation:-

P) df = pd.read_csv('iris.csv')
df.head()

O/P:-

	sepal_len	sepal_wid	pet_len	pet_wid	label
0	4.9	3.0	1.4	0.2	Iris-setosa
1					
2					
3					
4					

P) X = df.iloc[:, :-1] (: → all rows; all columns except last column)
y = df['label']
X.shape
y.shape

O/P:-
(149, 4) → data frame
(149,) → series.

P) 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)
print(x_train.shape)
x_test - 11
y_train. 11
y_test 11

O/P:- (111, 4)
(38, 4)
(111,)
(38,)

P) from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

P) m1 = DecisionTreeClassifier(criterion='gini', max_depth=3, min_samples_split=15)
m1.fit(x_train, y_train)

P) Print('Training score', m1.score(x_train, y_train))
Print('Testing score', m1.score(x_test, y_test))

O/P:-
Training score 0.981981
Testing 11 0.973684

P)
```
ypred_m1 = m1.predict(x_test)
print(ypred_m1)
```

O/P)

```
['Iris-versicolor', 'Iris-setosa'
 'Iris-virginica' ... ...
 - - - - ]
```

P) from sklearn.metrics import
confusion_matrix, classification_report

O/P)

P) def eval_metrics(ytest, ypred):
    cm = confusion_matrix(ytest, ypred)
    print(cm)
    print(classification_report(ytest, ypred))

P) eval_metrics(y_test, ypred_m1)

O/P)

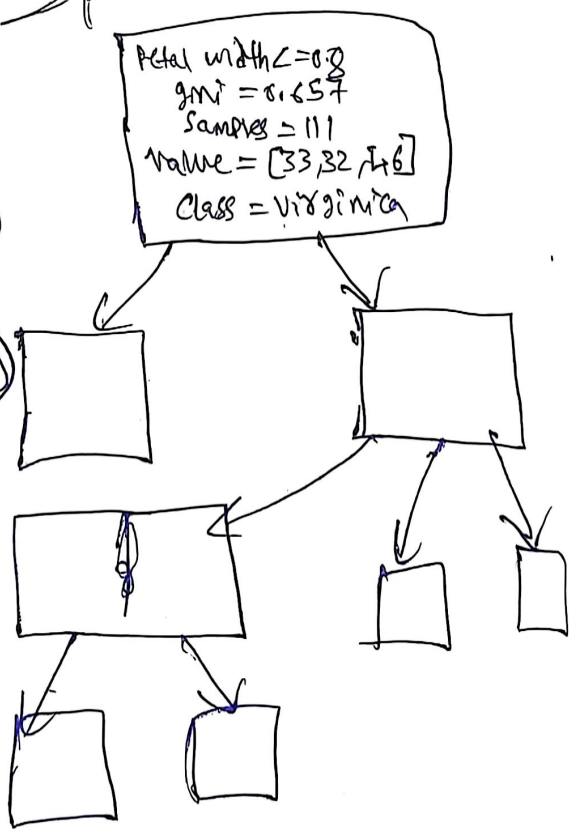
```
[[16  0  0
 [ 0 17  1
 [ 0  0  4]]
```

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	16
Iris-versicolor	1.00	0.94	0.97	18
Iris-virginica	0.80	1.00	0.89	4
accuracy			0.97	38

Plotting decision Tree

P) from sklearn.tree import plot_tree

```
fn = x_train.columns
cn = ['setosa', 'versicolor', 'virginica']
plot_tree(m1, feature_names = fn,
          class_names = cn, filled = T
plt.show()
```



SVM (Support Vector Machines):

→ A SVM is a supervised ML

→ A SVM takes in the data points (inputs) and outputs the hyperplane (which in two dimensions is simply a line) that best seperates the data points.

→ The objective of the SVM algorithm is to find a hyperplane in an N-dimensional space (N- the no. of independent features) that distinctly classifies the data points.

→ It follows a technique called the kernel trick to transform the data and based on these transformations, it finds an optimal boundary between the possible outputs

→ SVM objective - The main idea is to identify the optimal seperating hyperplane which maximizes the margin of the training data.

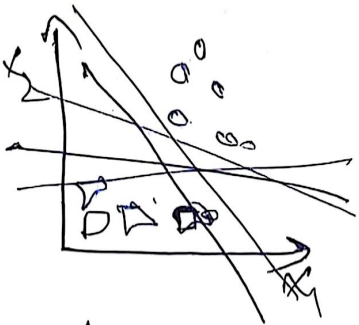
→ There can be multiple hyperplanes, but which one of them is the best seperating hyperplane?

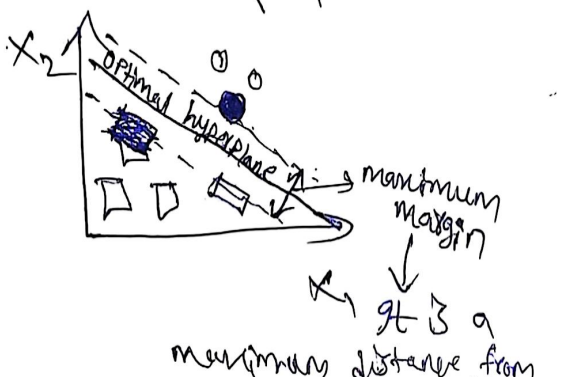
→ There can be multiple seperating hyperplanes as well, then we find the optimal one? Intuitively, if we select a hyperplane which is close to the data points of one class, then it might not generalize well (may not perform well on testing data).

→ So, the aim is to choose the hyperplane which is as far as possible from the data points of each category, Therefore maximizing the distance b/w the nearest points of each class and the hyperplane would result in an optimal seperating hyperplane,

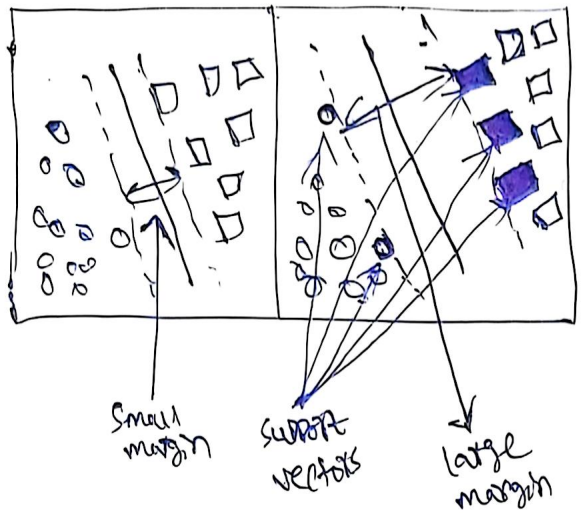
→ This distance is called margin, The goal of SVMs is to find the optimal hyperplane because it not only classifies the existing dataset but also helps predict the class of unseen data

→ And the optimal hyperplane is the one which has the biggest margin.





→ SVM's objective is to find a plane that has the maximum margin, i.e. the maximum distance b/w data points of both classes.



Using these support vectors, the model maximizes the margin of the classifier. These are the points that help us build our SVM.

3) Margin :-

It is the distance b/w support vectors and the hyperplane

→ A hyperplane in 2D is "line"

→ " " " 3D is "plane"

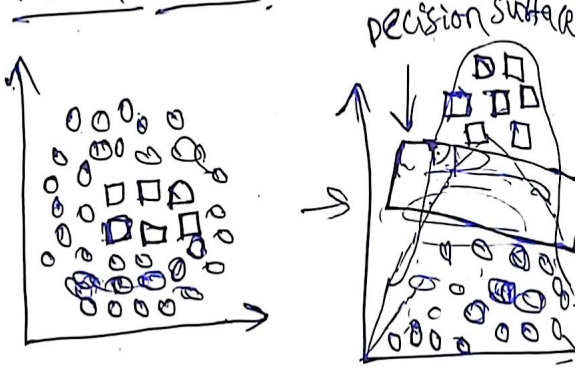
Terminologies

1) Hyperplane – Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. If the no. of input features is 2, then the hyperplane is just a line. If the no. of input features is 3, then the hyperplane becomes a two-dimensional plane. It becomes difficult when the no. of features exceeds 3.

2) Support vectors :-

These are data points that are closest to the hyperplane and influence the position and

Kernel Trick :-



SVM Parameters

1) Gamma :-

→ The gamma parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'.

If gamma is low ⇒ data points far away from hyper plane, data points affect the hyperplane.

If gamma is high ⇒ data points close to hyperplane, data "

Commonly used gamma values are [0.0001, 0.001, 0.01, 0.1, 1, 10, 100]

2) C :

→ The C parameter tells the SVM optimization how much you want to avoid misclassifying each training example. Small values of C attribute to more misclassification and large values of C attribute to less misclassification.

Commonly used C values are [0.0001, 0.001, 0.01, 0.1, 1, 10, 100]

3) Kernel :

SVM follows the technique called the kernel trick to transform the data and based on these transformations, it finds an optimal boundary between the possible outputs.

→ Different kernel options are linear kernel, radial basis function kernel (Rbf), polynomial kernel (Poly),

4) degree :

Defines degree of polynomial for 'Poly' kernel

Kernel Types :

1) Linear $K(x_i, x_j) = x_i \cdot x_j$. Here x_i, x_j represents data you are trying to classify.

2) Poly (polynomial) $K(x_i, x_j) = (x_i \cdot x_j + 1)^d$

Here "." shows dot product of both the values and d denotes the degree of the polynomial. $K(x_i, x_j)$ representing the decision boundary to seperate the given classes.

3) Rbf (Radial Basis function) :

$$K(x_i, x_j) = \exp(-\gamma * ||x_i - x_j||^2)$$

The value of $\gamma > 0$.

$$||\vec{x}|| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

SVM pros and cons!

pros !-

→ SVM is an algorithm which is suitable for both linearly and nonlinearly seperable data (using Kernel trick)

→ SVM is very good when we have no idea about the data.

→ Kernel trick is what makes SVM unique

Cons:-

→ They are not suitable for larger datasets because the training time with SVMs can be high and much more computationally intensive.

→ Tuning parameters can be time consuming

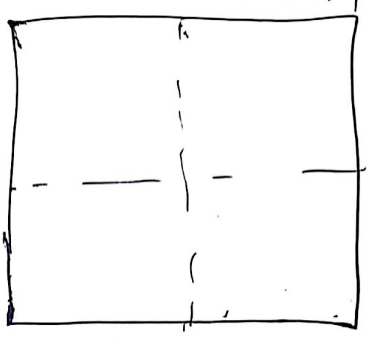
*** When we didn't know the data is linearly seperable or not, just blindly apply SVM or decision tree or Random forest, then there is a possibility to get good results.

SVM Simulator:-

link

https://jgreitemann.github.io/svm-demo/

After opening link interactive demo of SVM



Kernel: ▭ γ = ▭ . C =

K-Nearest Neighbors (KNN): Algo

→ It is supervised learning used for classification.

→ KNN Intuition - The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

→ KNN works by finding the distance between a query (test data) and all the examples in data (train data), selecting the specified no. of examples (K) closest to the query, then votes for the most frequent label (in the case of classification).

KNN Classification Algorithm:-

1) Load the data.

2) Initialize K to your chosen no. of neighbors.

3) For getting the predicted class, iterate from 1 to total no. of training data points.

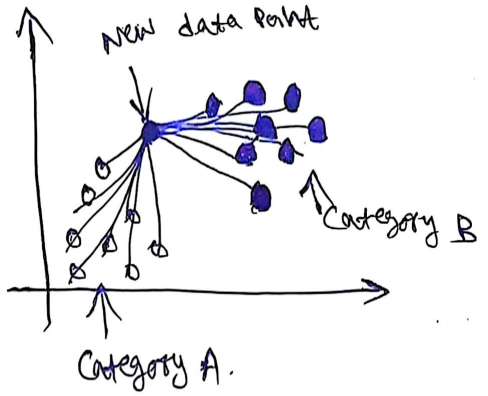
(a) Calculate the distance b/w test data and each row of training data. Use Euclidean distance as our distance metric since it's the most popular method. The other metrics that can be used are Manhattan, etc.

b) Sort the calculated distances in ascending order based on distance values.

c) Get top K rows from the sorted array.

d) Get the most frequent class (mode) of these rows.

e) Return the predicted class.



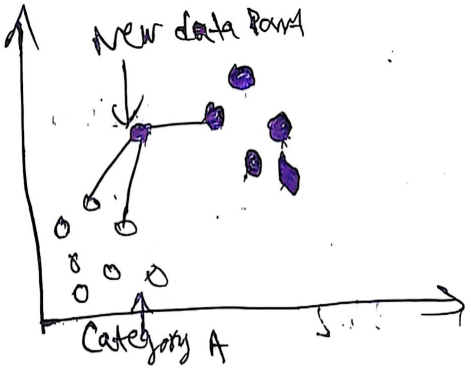
→ First compute distance from all data points

→ And then arrange in ascending order.

→ Got top K rows from sorted array.

→ Get the most frequent class (mode) of these rows as shown below.

→ Then return predicted class.



→ Then new data point belongs to category A.

Different distance metrics in K-NN

1) Euclidean distance:

A = (x_1, y_1) B = (x_2, y_2)

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

2) Manhattan distance:

$$AB = |x_2-x_1| + |y_2-y_1|$$

3) Minkowski's Distance:

$$AB = \left(|(x_2-x_1)^p| + |(y_2-y_1)^p|\right)^{1/p}$$

Note

for p=2, Minkowski's distance equal to Euclidean distance

for p=1, " " " to Manhattan distance.

Pros:

→ low computation time.

→ simple and easy to implement;

Cons:

→ It assumes that similar things occur in close proximity which can't be generalized for every single scenario.

Titanic survival prediction :-

P) import libraries.

P) df = pd.read_csv('titanic_train.csv')
df.head()

O/P :-

	PassengerId	Survived	...	Embarked
0				
1				
2				
3				
4				

P) df.shape

O/P :- (891, 12)

P) df['Survived'].value_counts()

O/P :-

0 549
1 342

Name: Survived, dtype: int64

* In general, whenever we proceed with model building we exclude the columns like PassengerId, roll no, registration number, name, certification, institution numb

P) df.isnull().sum()

O/P :-

PassengerId	0
Survived	0
Pclass	0
Name	0
Sex	0
Age	177
SibSp	0
Parch	0
Ticket	0
Fare	0
Cabin	687
Embarked	2

P) df.dtypes

O/P :-

PassengerId	int64
Survived	int64
Pclass	int64
Name	object
Sex	object
Age	float64
SibSp	int64
Parch	int64
Ticket	object
Fare	float64
Cabin	object
Embarked	object

dtype: object

Handling NULL values.

P) df['Age'].value_counts()

O/P:-

24.00	30
22.00	27
18.00	26
...	...
55.50	1
74.00	1

To compute % of null values per column.

P)
```
Y = pd.DataFrame(df.isnull().sum())
Y.columns = ['Count']
Y
```

O/P:

	Count
PassengerId	0
Survived	0
Pclass	0
Name	0
Sex	0
Age	177
SibSp	0
Parch	0
Ticket	0
Fare	0
Cabin	687
Embarked	2

P)
```
Y['Null_%age'] = Y['Count']*100 / Y['Count'].sum()
Y
```

O/P:

	Count	Null_%age
PassengerId	0	0.0000
Survived	0	0.0000
...	...	...
Age	177	20.438799
...	0	0.000
Cabin	687	79.330254
Embarked	2	0.230947

To Getrid of null values

P)
```
df['Age'].fillna(df['Age'].mean(), inplace=True)
df.isnull().sum()
```

O/P:

PassengerId	0
...	...
Age	0
...	...
Cabin	687
Embarked	2

dtype: int64

- Out of total rows how many values are null.

P) `df['%age_of_Null'] = df['Count']*100 / len(df)`

O/P :

	Count	Null%age	%age_of_null
PassengerId	0	0.00	0.00
Survived	0		
Age	177	20.438	19.88532
	0	0.00	0.00
Cabin	687	79.338254	77.104377
Embarked	2	0.2309	0.224467

Dropping non-significant columns

P) `df.drop(['PassengerId', 'Cabin', 'Name'], axis=1, inplace=True)`

↓

along columns

P) `df.isnull().sum()`

O/P :

Survived	0
Pclass	0
Sex	0
Age	
Embarked	0

P) `df['Embarked'].value_counts()`

O/P :

S	644
C	168
Q	77

Name: Embarked, dtype: int64

P) `df['Embarked'].fillna('S', inplace=True)`

`df.isnull().sum()`

O/P :

PassengerId	0
Age	0
Cabin	687
Embarked	0

Label Encoding

P) `df.dtypes`

O/P :

Survived	int64
Sex	object
Age	float64
Ticket	object
Fare	float64
Embarked	object

P) `df['Sex'].value_counts()`

O/P :

male	577
female	314

Name: Sex, dtype: int64

P) df['Ticket'].nunique()

O/P

681

It is irrelevant feature, drop it.

P) df.drop('Ticket', axis=1, inplace=True)

df.dtypes

O/P

Survived	int64
Pclass	int64
sex	object
Age	float64
⋮	⋮
Embarked	object

P) df.head()

O/P

	Survived	Pclass	Sex	Age	...	Embarked
0	^	^	male	|		S
1	^		female			C
2			female			S
3	(		female)			S
4	|		male			

Convert sex, Embarked into integer

Encoding categorical columns

sex	sex_lb	sex_M	sex_F
m	0	1	0
f	1	0	1
m	0	1	0
m	0	1	0
f	1	0	1
f	1	0	1
m	0	1	0

label encoding

This approach is One hot encoding

P) from sklearn.preprocessing import LabelEncoder

P) lb = LabelEncoder()

df['sex'] = lb.fit_transform(df['sex'])

df['Embarked'] = lb. " (df['Embarked'])

P) df['Embarked'].value_counts()

O/P

2	646	#S 646
0	168	#C 168
1	77	#Q 77

Name: Embarked, dtype: int64.

0,1,2 assing alphabetically.

P) df['sex'].value_counts()

O/P

1	577	#male 577
0	314	#female 314

Then :-

P) df.head()

O/P :-

	Survived	Sex	fare	Embarked
0		1		2
1		0		0
2		0		2
3		0		2
4		1		2

Building the models :-

P) df.columns

O/P :-

Index(['Survived', 'PClass', 'Sex', ...
..., 'Embarked'], dtype='object')

P) x = df.drop('survived', axis=1) #df.iloc[:, 1:] (All rows → columns starting from 1)

y = df['survived']

print(x.columns)

O/P :-

Index(['PClass', 'Sex', 'Age', ..., 'Embarked'], dtype='object')

P) print(type(x))
print(type(y))

O/P :-

<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>

P) 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)
print(x_train.shape)
print(x_test.shape)

```
print(y_train.shape)
print(y_test.shape)
```

O/P :-

```
(668, 7)
(223, 7)
(668,)
(223,)
```

```
from sklearn.linear_model import LogisticRegression.
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
```

```
from sklearn.metrics import confusion_matrix, classification_report
```

```
def eval_metrics(ytest, ypred):
    cm = confusion_matrix(ytest, ypred)
    print(cm)
    print(classification_report(ytest, ypred))
```

LogisticRegression :-

```
m1 = LogisticRegression(max_iter=1000)
m1.fit(x_train, y_train)
```

```
def mscore(model):
    print('Training score', model.score(x_train, y_train))
    print('Testing score', model.score(x_test, y_test))
```

p) mscore (m1)

O/P } Training score 0.80389
Testing score 0.7837 → Accuracy.

p) ypred_m1 = m1.predict (x_test).

p) eval metrics (y_test, ypred_m1)

	0	1
#0	[TP	FN]
#1	[FP	TN]

O/P }-

[[[112 18]
[28 65]]

	precision	recall	f1-score	support
0	0.80	0.86	0.83	130
1	0.78	0.70	0.74	93
accuracy			0.79	223
macro avg	0.79	0.78	0.78	223
weighted avg	0.79	0.79	0.79	223

#0 - no.of non-survivors, 1-no.of survivors.

#TP - Actual value is 0, ML model predicted 0 = 122

#FN - " " " 0, " " " 1 = 18

FP - " " " 1, " " " 0 = 28

#TN - " " " 1, " " " 1 = 65

TP=112 , FN = 18

FP=28 , TN =65

#preo = TP/(TP+FP) = 112/(112+28)

#pre1 = TN/(TN+FN) = 65/(65+18)

#reco = TP/(TP+FN) = 112/(112+18)

#rec1 = TN/(TN+FP) = 65/(65+28)

print ("preo", preo)

o/p)

preo 0.8

pre1 0.7831

reco 0.861

rec1 0.6989

Decision Tree

m2 = DecisionTreeClassifier(criterion='gini', max_depth=6, min_samples_split=15)

m2.fit(x_train, y_train)

mscore(m2)

o/p) Training score 0.8802

Testing score 0.81165

p) ypred_m2 = m2.predict(X_test)

ypred_m2

o/p) array([1,0,0,---

- - - - -])

p)

eval_metrics(y_test, ypred_m2)

o/p)

[[118 12

[30 63]]

	precision	recall	f1-score	support
0	0.80	0.91	0.85	130
1	0.84	0.68	0.75	93
accuracy			0.81	223
macro avg	0.82	0.79	0.80	223
weighted avg	0.82	0.81	0.81	223

Random Forest

m3 = RandomForestClassifier(n_estimators=90, criterion='entropy', min_samples_split=20, max_depth=8)

m3.fit(X_train, Y_train)

p) mscore(m3)

o/p) Training score 0.8787

Testing score 0.8206

p) ypred_m3 = m3.predict(x_test)

p) eval_metrics(y_test, ypred_m3)

O/P :-

[[123 7]
 [33 60]]

	precision	recall	f1-score	support
0	0.79	0.95	0.86	130
1	0.90	0.65	0.75	93
accuracy			0.82	223
macro avg	0.84	0.80	0.81	223
weighted avg	0.83	0.82	0.81	223

KNN Classifier :-

P) m4 = KNeighborsClassifier(n_neighbors = 41)
(This number should be odd)

m4.fit(x_train, y_train)

P) mscore(m4)

O/P :-

Training score 0.71556
Testing " 0.6816

P) ypred_m4 = m4.predict(x_test)

P) eval_metrics (y_test, ypred_m4)

O/P :- [[112 18]
 [53 40]].

	precision	recall	f1-score	support
0	0.68	0.86	0.76	130
1	0.69	0.43	0.53	93
accuracy			0.68	223
macro avg	0.68	0.65	0.64	223
weighted avg	0.68	0.68	0.66	223

SVM :-

P) m5 = SVC(kernel='linear', C=1)
m5.fit (x_train, y_train)

P) mscore(m5)

O/P :-

Training score 0.7889
Testing score 0.7802

P) ypred_m5 = m5.predict(x_test)
print(ypred_m5)

O/P :-

[1 1 0 0 - - - - - - - - - - - - -]

P) eval_metrics(y_test, ypred_m5)

O/P :-

[[110 20]
 [29 64]]

	precision	recall	f1-score	support
0	0.79	0.85	0.82	130
1	0.76	0.69	0.72	93
accuracy			0.78	223
macro avg	0.78	0.77	0.77	223
weighted avg	0.78	0.78	0.78	223

So, Best performing model is RandomForestClassifier.

Gender wise patient survival count

```
p) y1 = df.groupby(['Survived', 'Sex'])['Sex'].count()
```

op):

Survived	Sex	
0	0	81
	1	468
1	0	233
	1	109

Name: Sex, dtype: int64.

* If training score is high when compared to testing score, the the model is overfitted

for example)-

Training score 0.899
Testing score 0.744

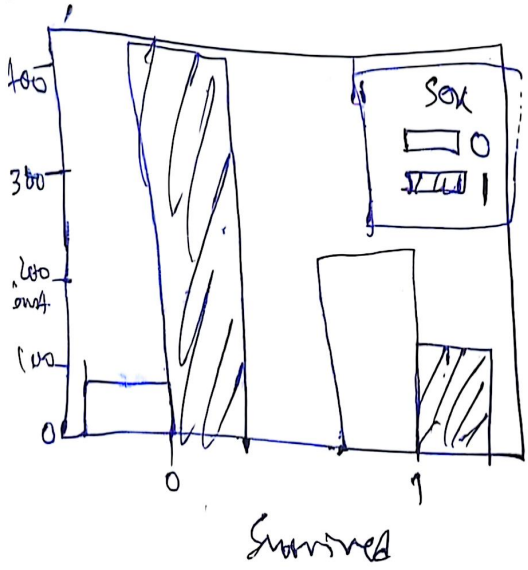
↙

then this model is over fit model

we can use functions to write all the common things.

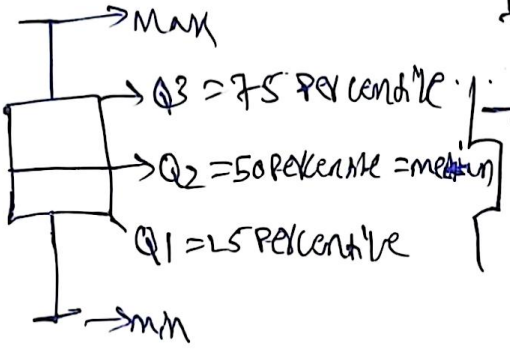
```
p) import seaborn as sns
sns.countplot(x=df['Survived'], hue=df['Sex'])
plt.show()
```

op):




```
p) def model_gen(model, x_train, x_test, y_train, y_test):
    model.fit(x_train, y_train)
    print('Training score', model.score(x_train, y_train))
    print('Testing score', model.score(x_test, y_test))
    ypred = model.predict(x_test)
    print(ypred)
    cm = confusion_matrix(y_test, ypred)
    print('Confusion_Matrix\n', cm)
    print('Classification Report\n', classification_report(y_test, ypred))
```

About boxplot



IQR = Inter Quartile range

= Q3 − Q1

- max = Q3 + 1.5 * IQR
  min = Q1 − 1.5 * IQR

→ Any values that are greater than max and less than min are called outliers.

Clustering

→ Clustering identifies similarities between objects, which it groups according to those characteristics in common and which differentiate them from other groups of objects. These groups are known as "clusters".

→ Clustering is framed in unsupervised learning; that is, for this type of algorithm we only have one set of input

without previously knowing what the output will be.

→ There is no need to split the data in training and testing dataset.

KMeans

→ K-means algorithm is an iterative algorithm that tries to partition the dataset into K pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group.

→ It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns a data points to a cluster such that the sum of the squared distance between the data points and the cluster centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation

more homogeneous (similar) the data points are within the same cluster.

⇒ since clustering algorithms including KMeans which use distance based measurements to determine the similarity between data points, its recommended to standardize or scale the data since almost always the features in any dataset would have different units of measurements for instance as age vs. income.

K-means Algorithm:

1) specify no. of clusters
2) Initialize centroids by first shuffling the dataset and then randomly selecting K data points for the centroids without replacement.
3) Keep iterating until there is no change to the centroids. i.e. assignment of data points to cluster isn't changing.
4) Compute the sum of the squared distance b/w data points and all centroids.
5) Assign each data point to the closest cluster (centroid)
6) Compute the centroids for the clusters by taking the average of the all data points that belong to each cluster.

Elbow method to find the optimal number of clusters:

→ It gives us an idea on what a good K no. of clusters would be based on the WCSS (within cluster sum of square) between data points and their assigned clusters centroid. We pick 'K' at the spot where WCSS starts to flatten out and forms an elbow

WCSS = sum of squared distance b/w individual data points and its closest centroid for all the data points.

Pros:
→ Simple to implement & understand.
→ It gives best results, when data sets are distinct.

Cons:
→ The user has to specify K (no. of clusters in the beginning)

→ Data is required to be scaled in most cases.

→ Centroids can be dragged by the outliers.

→ As the no. of dimensions (no. of columns/features) increases, it is recommended us to use PCA to reduce dimensions.

P) df.head()

O/P

	CID	Gender	Age	...	Sp
0	1		19		
1	2		21		
2	3				
3	4				
4	5				

P) df.iloc[:, -2:] = df

df.head()

O/P

	AnnualIncome	SpendingScore
0	15	39
1	15	81
2	16	6
3	16	77
4	17	40

P) df = pd.read_csv('Customers.csv')

df.head()

O/P

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2				81
2	3				
3	4				
4	5				

P) X = df.values

X[:5]

O/P

array([[15, 39],
[15, 81],
[16, 6],
[16, 77],
[17, 40], dtype=int64)

Renaming the columns.

P) df.columns = ['CID', 'Gender', 'Age', 'AnnualIncome', 'SpendingScore']

df.columns

O/P

Index(['CID', 'Gender', ...], dtype='object')

Cluster AnnualIncome and SpendingScore!

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

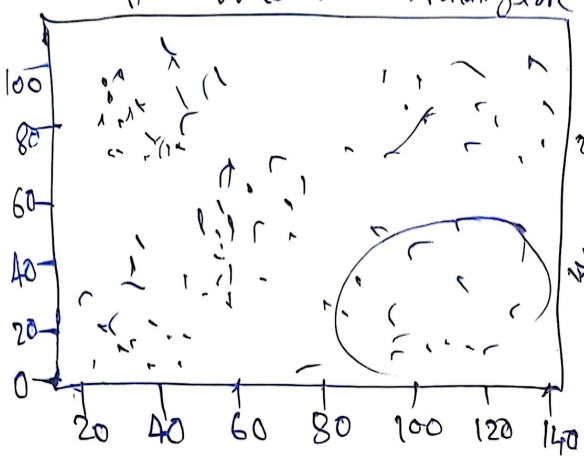
plt.title('AnnualIncome vs SpendingScore')

plt.xlabel('AnnualIncome')

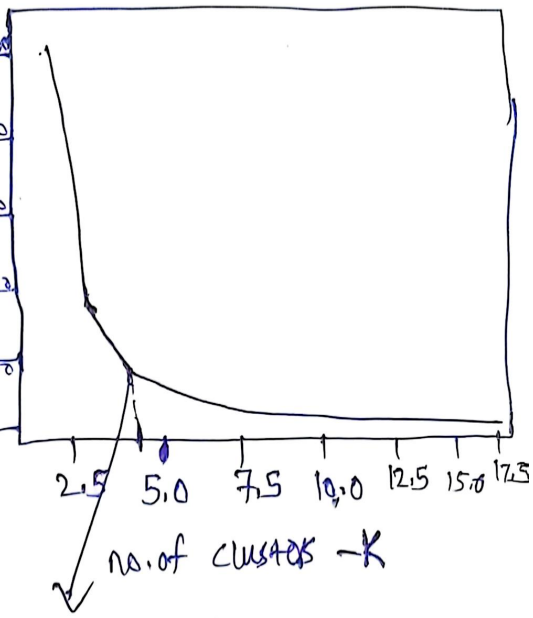
plt.ylabel('SpendingScore')

plt.show()

O/P)



O/P)



Elbow method

The bend is near 5.0

Then it is optimal no. of clusters.

P)
```
from sklearn.cluster import
                    KMeans
```

P)
```
wcss=[]
for i in range(1,19):
    m=KMeans(n_clusters=i)
    m.fit(X)
    wcss.append(m.inertia_)

print('wcss\n', wcss)
plt.plot(list(range(1,19)), wcss)
plt.xlabel('num of clusters-K')
plt.ylabel('WCSS')
plt.title('Elbow Method')
plt.show()
```

O/P
```
WCSS
[269981.28, 181363.5959,
 106348.37306 ... ]
```

Applying KMeans at K=5

P)
```
KS=KMeans(n_clusters=5)
KS.fit(X)
```

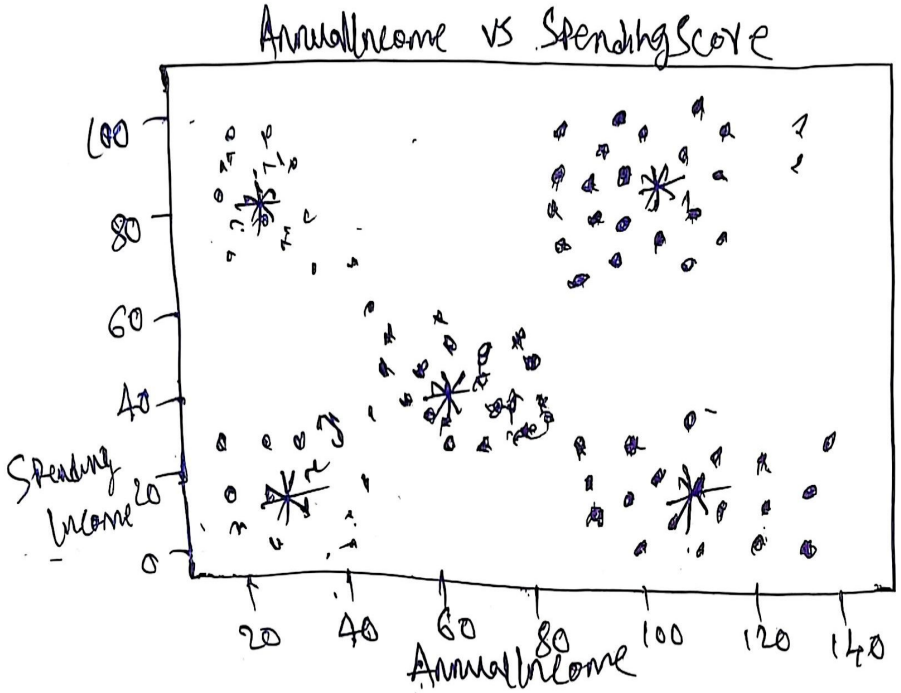
P)
```
ypred5=KS.predict(X)
ypred5
```

O/P
```
array([4,1,4,1,4,1 ...
       ..... 2, ...
       ... ,0, ... ,3])
```

P)
```
cent5=KS.cluster_centers_
cent5
```

O/P
```
array([[88.2,17.114285 7],
       [25.72273,79.36363 6],
       [ ... ]
```

P)
```
plt.scatter(X[:,0], X[:,1], c=ypred5)
plt.scatter(cent5[:,0], cent5[:,1], color='red', s=130, marker='*')
plt.title('AnnualIncome vs SpendingScore')
plt.xlabel('AnnualIncome')
plt.ylabel('SpendingScore')
plt.show()
```



Aplying kmeans at K=3

P)
```
k3 = KMeans(n_clusters=3)
k3.fit(X)
```

P)
```
ypred3 = k3.predict(X)
ypred3
```

O/P) array([0,0, ... ,2,1,2,1, ... ,2,1,2,1,2])

P)
```
cent3 = k3.cluster_centers_
print(cent3)
```

O/P)
```
[[44.1544   49.8292]
 [87.       18.6315]
 [86.5384   82.1282]]
```

P) plt.scatter (X[:,0], X[:,1], c=ypred3)

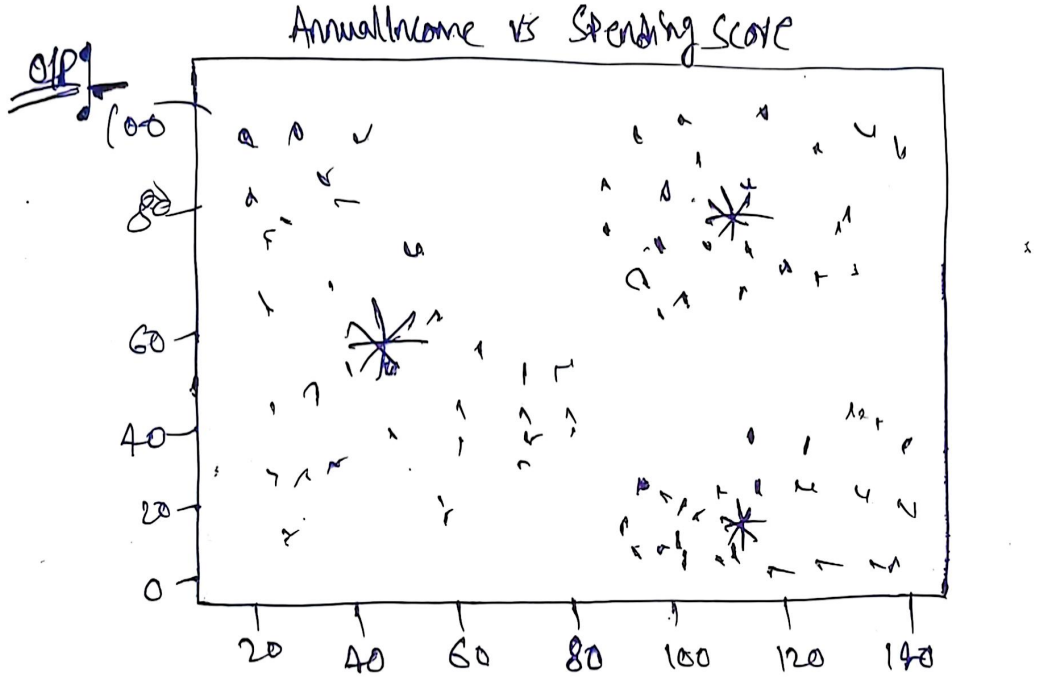
plt.scatter (cent3[:,0], cent3[:,1], color='red', s=150, marker='*')

plt.title('AnnualIncome vs SpendingScore')

plt.xlabel('AnnualIncome')

plt.ylabel('SpendingScore')

plt.show()



Mean-Shift Clustering Algorithm

→ We don't have to specify the no.of clusters at the time of model Creation.

→ It is based on a bandwidth parameter.

KMeans

→ We have to specify the no.of clusters at the time of model creation.

→ It assigns data points to a cluster such that the sum of the squared distance b/w the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum.

Hyperparameters in different ML models:

→ Randomforest (n_estimators, max_depth, criterion, min_samples_split)

→ DecisionTreeclassifier (max_depth, criterion, min_samples_split)

→ SVM (kernel, degree, C, gamma)

→ KNN (n_neighbors)

P) import libraries

P) df = pd.read_csv ('sonar.csv', header = None)
df.head()

O/P

	0	1	2	3	----	60
0	0.0200					R
1						R
2						R
3						M
4						M

P) df.shape

O/P (208, 61)

P) df.dtypes.value_counts()

O/P

float64	60
object	1

P) nv = df.isnull().sum()
rl = nv[nv>0]
rl

O/P - Series ([], dtype: int64)

P) x = df.iloc[:, :-1]
y = df.iloc[:, -1]
print(x.shape)
print(y.shape)

O/P (208, 60)
(208,)

P) df[60].value_counts()
#M - Mines
#R - Rocks

O/P

M	111
R	97

P) from sklearn.model_selection import train_test_split

P) x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25)

```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

O/P)

(156, 60)
(52, 60)
(156,)
(52)

P) from sklearn.metrics import confusion_matrix, classification_report.

P) def mscore(model):

```
    print('Training score', model.score(x_train, y_train))
    print('Testing score', model.score(x_test, y_test))
```

```
def gen_metrics(y_test, y_pred):
    cm = confusion_matrix(y_test, y_pred)
    print(cm)
    print(classification_report(y_test, y_pred))
```

Decision Tree

P) from sklearn.tree import DecisionTreeClassifier

P) m1 = DecisionTreeClassifier(criterion='gini', max_depth=9, min_samples_split = 15)

m1.fit(x_train, y_train)

P) mscore(m1)

O/P)

Training score 0.9230
Testing score 0.6730

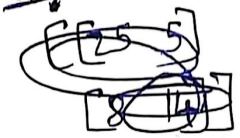
P) ypred_m1 = m1.predict(x_test)
print(ypred_m1)

O/P)

['R' 'M' 'M' 'R' - -
- - - -]

P) gen_metrics(y_test, ypred_m1)

O/P)



[[24 5
[12 11]]

	precision	recall	f1-score	support
M	0.67	0.83	0.74	29
R	0.69	0.48	0.56	23
accuracy			0.67	52
macro avg	0.68	0.65	0.65	52
weighted avg	0.68	0.67	0.66	52

Hyperparameter Tuning

→ Tuning model hyperparameters such that the model generates the best performance.

There are 2 methods of Hyperparameter Tuning.

1) GridSearchCV

→ It takes into consideration all the permutations & combinations of the hyperparameters & returns the hyperparameters that would generate the best result.

→ It consumes a lot of time.

2) RandomizedSearchCV

→ It takes into consideration some random subset of hyperparameters and return the hyperparameters that would generate the best result out of those randomly sampled hyperparameters.

→ It consumes a comparatively less time.

P) from sklearn.model_selection import RandomizedSearchCV, GridSearchCV

GridSearchCV on DecisionTree

P)
```
params_df = {'criterion': ['gini', 'entropy'],
             'max_depth': list(range(7,12)),
             'min_samples_split': [10,12,15,17,18,20]}
```

```
params_df
```

O/P)
```
{'criterion': ['gini', 'entropy'],
 'max_depth': [7, 8, 9, 10, 11],
 'min_samples_split': [10, 12, 15, 17, 18, 20]}
```

P)
```
dtl = DecisionTreeClassifier()
gs1 = GridSearchCV(estimator=dtl, param_grid=params_df, scoring='accuracy')
gs1.fit(x_train, y_train)
```

O/P)

▸ GridSearchCV
▸ estimator: DecisionTreeClassifier

P) print(gs1.best_params_)

print(gs1.best_estimator_)

O/P)

{'criterion': 'gini', 'max_depth': 8,
'min_samples_split': 12}

DecisionTreeClassifier(max_depth=8,
min_samples_split=12)

P) ypred_gs1 = gs1.predict(x_test)

print(ypred_gs1)

O/P)-

['M' 'M' 'R' ...
...]

P) gen_metrics(y_test, ypred_gs1)

O/P)

[[25 4]
[12 11]]

	Precision	recall	f1-score	support
M	0.68	0.86	0.76	29
R	0.73	0.48	0.58	23
accuracy			0.69	52
macro avg	0.70	0.67	0.67	52
weighted avg	0.70	0.69	0.68	52

RandomizedSearchCV on Random Forest

* RandomForest & SVM takes more time when GridSearchCV is used. That's why RandomizedSearchCV is used for both RF & SVM.

P) from sklearn.ensemble import RandomForestClassifier

P) params_rf = {'n_estimators': [70, 80, 100, 120],
'criterion': ['gini', 'entropy'],
'max_depth': list(range(7, 12)),
'min_samples_split': [10, 12, 15, 17, 18, 20]}

P) rf1 = RandomForestClassifier()

rs1 = RandomizedSearchCV(estimator=rf1,
param_distributions=params_rf, scoring=
'accuracy', cv=8)

rs1.fit(X_train, Y_train)

O/P)

▶ RandomizedSearchCV

▶ estimator: RandomForestClassifier

P) print(rs1.best_params_)
print(rs1.best_estimator_)

O/P:-

{'n_estimators': 100, 'min_samples_split': 10, 'max_depth': 9, 'criterion': 'gini'}

RandomForestClassifier(max_depth=9, min_samples_split=10)

P) ypred_rs1 = rs1.predict(x_test)
print(ypred_rs1)

O/P:-

['R' 'M' 'R' 'R' 'R' - - - - -]

P) gen_metrics(y_test, ypred_rs1)

O/P:-

[[25 4]
[4 19]]

	precision	recall	f1-score	support
M	0.86	0.86	0.86	29
R	0.83	0.83	0.83	23
accuracy			0.85	52
macro avg	0.84	0.84	0.84	52
weighted avg	0.85	0.85	0.85	52

Applying GridSearchCV on KNN

P) from sklearn.neighbors import KNeighborsClassifier

P)
params_knn = {'n_neighbors': list(range(5,29))}
print(params_knn)

O/P)

{'n_neighbors': [5,6,7,8, - - - - - 27,28]}

P) knn1 = KNeighborsClassifier()
gs2 = GridSearchCV(knn1, param_grid = params_knn, scoring='accuracy', cv=10)
gs2.fit(x_train, y_train)

O/P:-

▸ GridSearchCV:
▸estimator: KNeighborsClassifier

P) print(gs2.best_params_)
print(gs2.best_estimator_)

O/P:-

{'n_neighbors': 5}
KNeighborsClassifier()

f) `ypred_gs2 = gs2.predict(x_test)`

`print(ypred_gs2)`

O/P)-

```
['R' 'M' 'M' 'M' ...

 ...                ]
```

g) `gen_metrics(y_test, ypred_gs2)`

O/P)-

```
[[26  3]
 [ 8 15]]
```

	precision	recall	f1-score	support
M	0.76	0.90	0.83	29
R	0.83	0.65	0.73	23
accuracy			0.79	52
macro avg	0.80	0.77	0.78	52
weighted avg	0.80	0.79	0.78	52