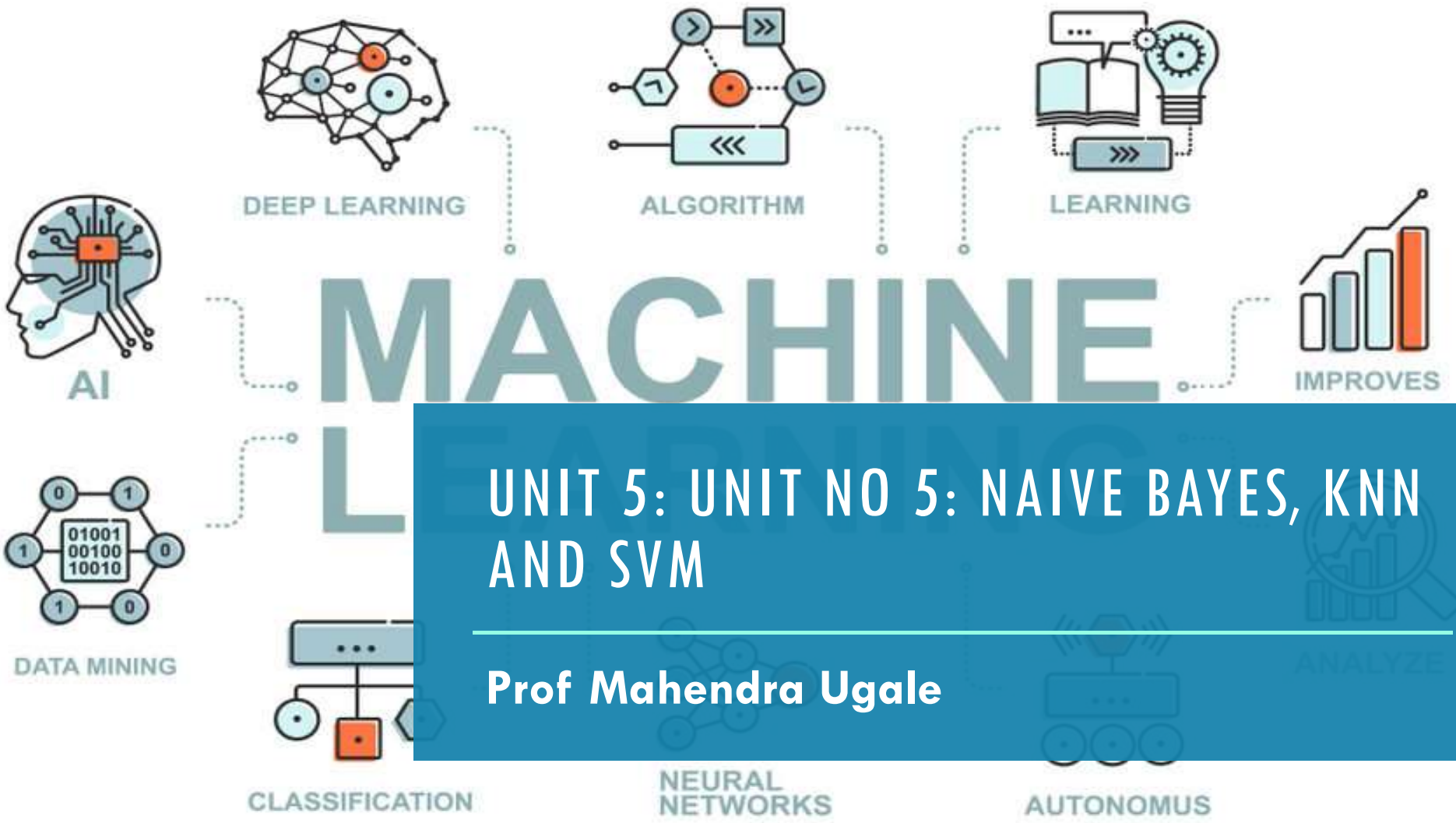
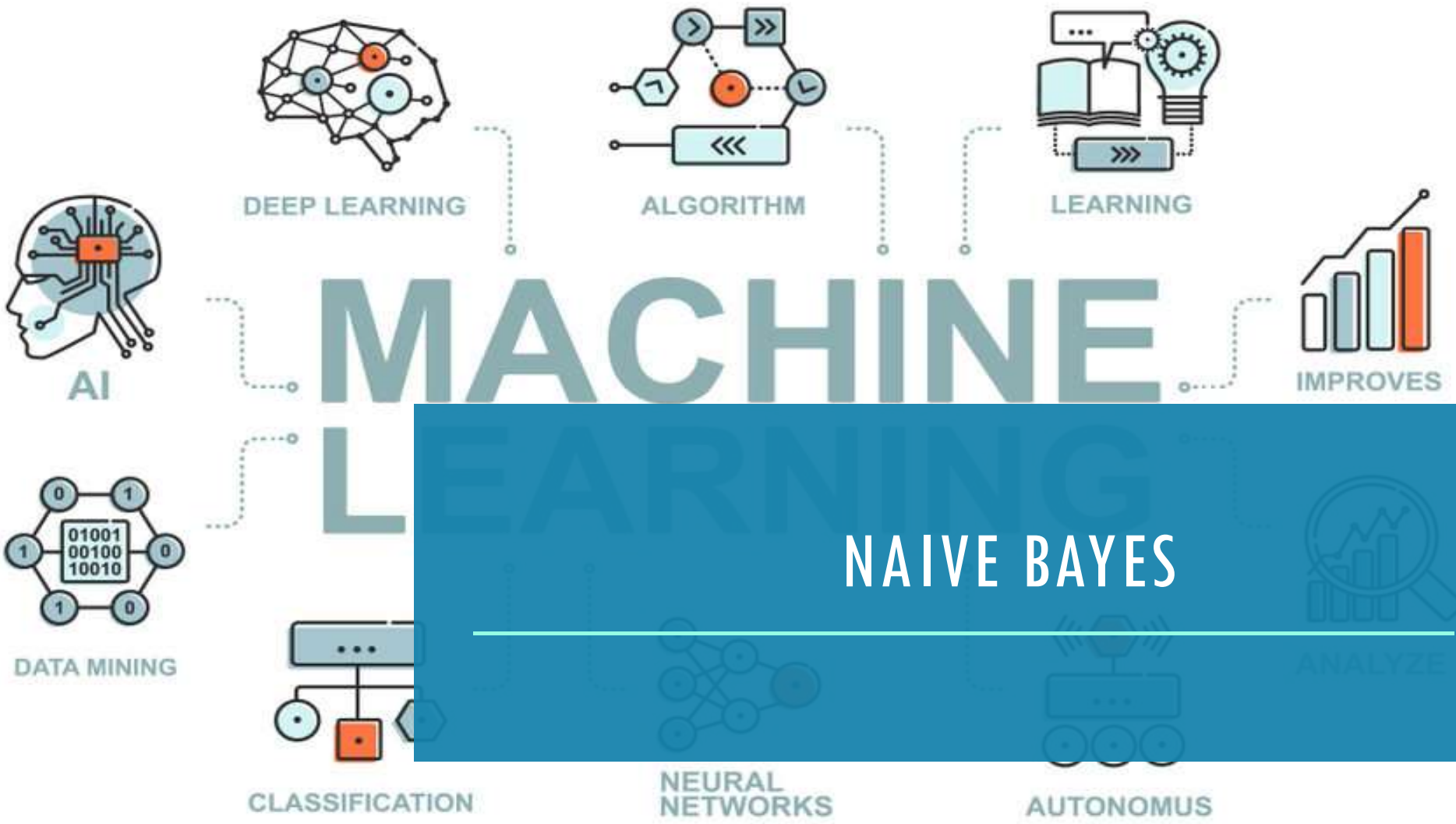


UNIT 5: UNIT NO 5: NAIVE BAYES, KNN AND SVM

Prof Mahendra Ugale


DEEP LEARNING
ALGORITHM
LEARNING
AI
MACHINE
IMPROVES
LEARNING
01001
00100
10010
DATA MINING
ANALYZE
CLASSIFICATION
NEURAL
NETWORKS
AUTONOMUS

NAIVE BAYES

Naïve Bayes Classifier

07 December 2022 12:53 PM

↳ Used for classification

↳ Based on probability → conditional probability

① Independent Events;

e.g: ① Rolling a dice: Sample space {1, 2, 3, 4, 5, 6}

$Pr(1) = 1/6$, $Pr(2) = 1/6$, $Pr(3) = 1/6$ $Pr(4) = 1/6$

$Pr(5) = 1/6$ $Pr(6) = 1/6$

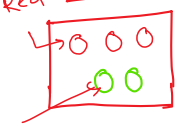
② Tossing a coin

$Pr(H) = 1/2$ $Pr(T) = 1/2$

---

② Dependent Events: →

Red



→ $Pr(R) = \frac{3}{5}$ → $Pr(G/R) = \frac{2}{4}$

↑ Event already performed

Green marble

$\{Pr(R \text{ and } G) = Pr(R) * P(G/R)\}$ → Conditional probability

$\{Pr(A \text{ and } B) = Pr(A) * P(B/A)\}$

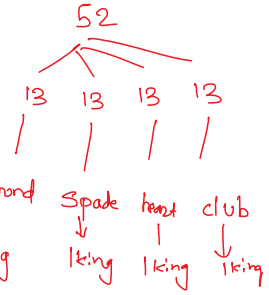
↳ Naive bayes classifier is based on Baye's theorem

Baye's theorem:

$$P(B/A) = \frac{Pr(B) * Pr(A/B)}{Pr(A)}$$

Prior → $Pr(B)$; likelihood → $Pr(A/B)$; Marginal → $Pr(A)$

Posterior probability → $P(B/A)$



---

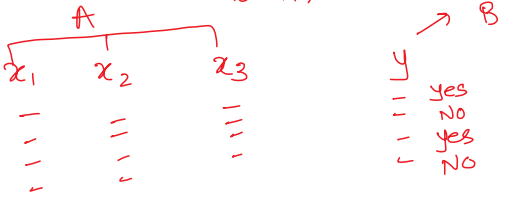
e.g: $P(\underset{B}{\text{King}} / \underset{A}{\text{face}})$

$$P(\text{king}/\text{face}) = \frac{Pr(\text{king}) * Pr(\text{face}/\text{king})}{Pr(\text{face})}$$

$$= \frac{4/52 * 1}{12/52} = 1/3$$

* How to Apply Naive Baye's to Dataset {

$$P(B/A) = \frac{Pr(B) * Pr(A/B)}{Pr(A)}$$ → Baye's theorem



$$P(y/x_1, x_2, x_3) = \frac{Pr(y) * P(x_1, x_2, x_3/y)}{\quad}$$

$$P(y \mid x_1, x_2, x_3) = \frac{P(y) * P(x_1\ x_2\ x_3 \mid y)}{P(x_1\ x_2\ x_3)}$$

e.g.:

x_1	x_2	x_3	x_4	o/p
–	–	–	–	NO ✓
–	–	–	–	yes ✓

General eqn

$$P(y \mid x_1, x_2, x_3 \ldots x_n) = \frac{P(y) * P(x_1\ x_2\ x_3 \ldots x_n \mid y)}{P(x_1\ x_2 \ldots x_n)}$$

for class yes

$$P(y \mid x_1, x_2, x_3 \ldots x_n) = \frac{P(y) * P(x_1 \mid y) * P(x_2 \mid y) * P(x_3 \mid y) * \ldots P(x_n \mid y)}{P(x_1, x_2 \ldots x_n)}$$

Constant → eqn ①

for class No:

$$P(N \mid x_1, x_2, x_3 \ldots x_n) = \frac{P(n) * P(x_1 \mid n) * P(x_2 \mid n) * P(x_3 \mid n) * \ldots P(x_n \mid n)}{P(x_1, x_2 \ldots x_n)}$$

Constant → eqn ②

New Data

x_1 x_2 x_3 x_4 → Y

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

feature No: 1 — Outlook — P(sunny|yes)

Outlook	Yes	NO	P(y)	P(N)
Sunny	2	3	2/9	3/5
overcast	4	0	4/9	0/5
Rain	3	2	3/9	2/5
	9	5	≈ 14	

P(sunny|NO)

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes

feature No: 2 — Temperature — P(Hot|yes)

Temperature	Y	N	P(Y)	P(N)
Hot	2	2	2/9	2/5
Mild	4	2	4/9	2/5
Cool	3	1	3/9	1/5

D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Mild	4	2	4/9	2/5
Cool	3	1	3/9	1/5
	9	5		

Output Column – "play Tennis"

	P(Y)	P(N)
Yes – 9	9/14	–
No – 5	–	5/14

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

New Data:

$P(\underset{y}{\text{yes}} \mid \underset{x_1}{\text{Sunny}}, \underset{x_2}{\text{Hot}})$

$P(\underset{y}{\text{No}} \mid \underset{x_1}{\text{Sunny}}, \underset{x_2}{\text{Hot}})$

Naive Bayes

$$P(B/A) = \frac{P(B) * P(A/B)}{P(A)}$$

$\rightarrow P(\underset{B}{\text{yes}} \mid \underset{A}{\text{Sunny, hot}}) = P(\text{yes}) * P(\text{Sunny} \mid \text{yes}) * P(\text{hot} \mid \text{yes}) \rightarrow (1)$

$\rightarrow P(\underset{B}{\text{No}} \mid \underset{A}{\text{Sunny, hot}}) = P(\text{No}) * P(\text{Sunny} \mid \text{No}) * P(\text{hot} \mid \text{No}) \rightarrow (2)$

Solve eqn (1)

$P(\text{yes} \mid \text{Sunny, hot}) = \frac{9}{14} * \frac{2}{9} * \frac{2}{9} \simeq 0.031$

Solve eqn (2)

$P(\text{No} \mid \text{Sunny, hot}) = \frac{5}{14} * \frac{3}{5} * \frac{2}{5} = 0.085$

$$P(\text{yes} \mid \text{Sunny, hot}) = \frac{0.031}{0.031 + 0.085} = 27\%$$

$$P(\text{No} \mid \text{Sunny, hot}) = \frac{0.085}{0.031 + 0.085} = 73\%$$

$P(\text{No} \mid \text{Sunny, hot}) > P(\text{yes} \mid \text{Sunny, hot})$

i.e. output belongs to "No" class.

Imortant Note About Random State-

In [12]:

```
import pandas as pd
df = pd.read_csv('http://bit.ly/kaggletrain', nrows=6)
df
```

Out[12]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q

In [14]:

```
cols = ['Fare', 'Embarked', 'Sex']
X = df[cols]
y = df['Survived']
```

In [15]:

```
X
```

Out[15]:

	Fare	Embarked	Sex
0	7.2500	S	male
1	71.2833	C	female
2	7.9250	S	female
3	53.1000	S	female
4	8.0500	S	male
5	8.4583	Q	male

In [16]:

```
y
```

Out[16]:

```
0    0
1    1
2    1
3    1
4    0
5    0
Name: Survived, dtype: int64
```

In [28]:

```
# any positive integer can be used for the random_state value

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)

# With random_state=0, We get the same train and test sets across different executions.

X_train
```

Out[28]:

	Fare	Embarked	Sex
3	53.10	S	female
0	7.25	S	male
4	8.05	S	male

In [27]:

```
X_train
```

Out[27]:

	Fare	Embarked	Sex
3	53.10	S	female
0	7.25	S	male
4	8.05	S	male

Naive Bayes Classifier

Naïve Bayes Classifier Algorithm

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.
- It is mainly used in text classification that includes a high-dimensional training dataset.
- 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.
- It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.
- 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:

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

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.

Problem:

If the weather is sunny, then the Player should play or not?

Solution:

To solve this, first consider the below dataset:

	Outlook	Play
0	Rainy	Yes
1	Sunny	Yes
2	Overcast	Yes
3	Overcast	Yes
4	Sunny	No
5	Rainy	Yes
6	Sunny	Yes
7	Overcast	Yes
8	Rainy	No
9	Sunny	No
10	Sunny	Yes
11	Rainy	No
12	Overcast	Yes
13	Overcast	Yes

Frequency table for the Weather Conditions:

Frequency table for the Weather Conditions:

Weather	Yes	No
Overcast	5	0
Rainy	2	2
Sunny	3	2
Total	10	5

Likelihood table weather condition:

Weather	No	Yes	
Overcast	0	5	5/14= 0.35
Rainy	2	2	4/14=0.29
Sunny	2	3	5/14=0.35
All	4/14=0.29	10/14=0.71	

Applying Bayes'theorem:

P(Yes|Sunny)= P(Sunny|Yes)*P(Yes)/P(Sunny)

P(Sunny|Yes)= 3/10= 0.3

P(Sunny)= 0.35

P(Yes)=0.71

So P(Yes|Sunny) = 0.3*0.71/0.35= 0.60

P(No|Sunny)= P(Sunny|No)*P(No)/P(Sunny)

P(Sunny|NO)= 2/4=0.5

P(No)= 0.29

P(Sunny)= 0.35

So P(No|Sunny)= 0.5*0.29/0.35 = 0.41

So as we can see from the above calculation that P(Yes|Sunny)>P(No|Sunny)

Hence on a Sunny day, Player can play the game.

Python Implementation of the Naïve Bayes algorithm:

Now we will implement a Naive Bayes Algorithm using Python.

So for this, we will use the "user_data" dataset, which we have used in our other classification model.

Therefore we can easily compare the Naive Bayes model with the other models.

Steps to implement:

Data Pre-processing step

Fitting Naive Bayes to the Training set

Predicting the test result

Test accuracy of the result(Creation of Confusion matrix)

Visualizing the test set result.

1) Data Pre-processing step:

In this step, we will pre-process/prepare the data so that we can use it efficiently in our code.

It is similar as we did in data-pre-processing

. The code for this is given below:

In [1]:

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

In [2]:

```
# Importing the dataset
dataset = pd.read_csv('user_data.csv')
dataset.head()
```

Out[2]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

In [3]:

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

In [4]:

```
# 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)
```

In [5]:

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

2) Fitting Naive Bayes to the Training Set:

After the pre-processing step, now we will fit the Naive Bayes model to the Training set.

Below is the code for it:

In [7]:

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

Out[7]:

```
GaussianNB()
```

In [ ]:

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.

3) Prediction of the test set result:

Now we will predict the test set result.

For this, we will create a new predictor variable y_pred, and will use the predict function to make the predictions.

In [8]:

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

4) Creating Confusion Matrix:

Now we will check the accuracy of the Naive Bayes classifier using the Confusion matrix.

Below is the code for it:

In [10]:

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

Out[10]:

```
array([[65,  3],
       [ 7, 25]], dtype=int64)
```

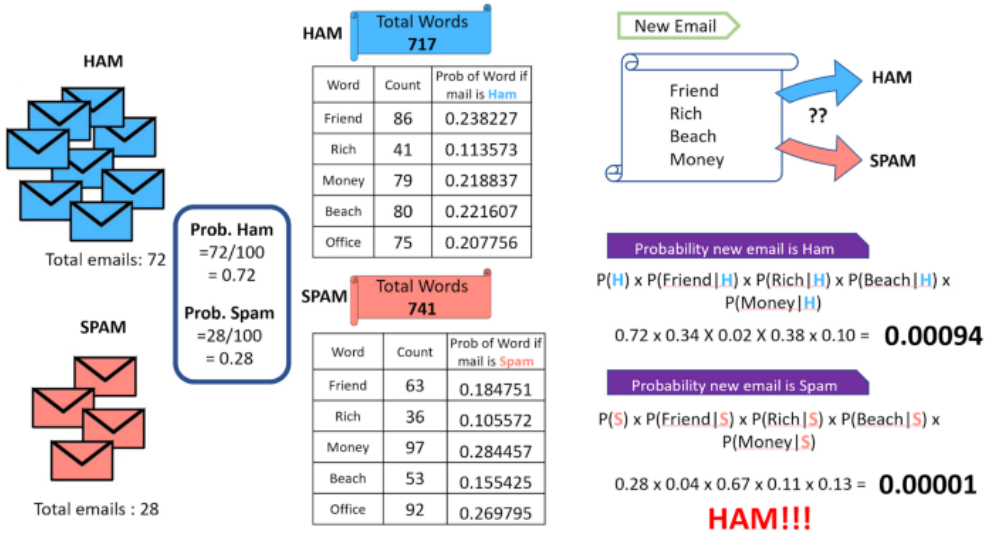
As we can see in the above confusion matrix output, there are 7+3= 10 incorrect predictions, and 65+25=90 correct predictions.

How to handle zero probabilities

Example — Let us understand this with an example of email classification as spam or ham (i.e. no spam).

We simply count the number of words in both classes of email and then find the probability of each word's probability given the class prior probability of that email as spam or ham.

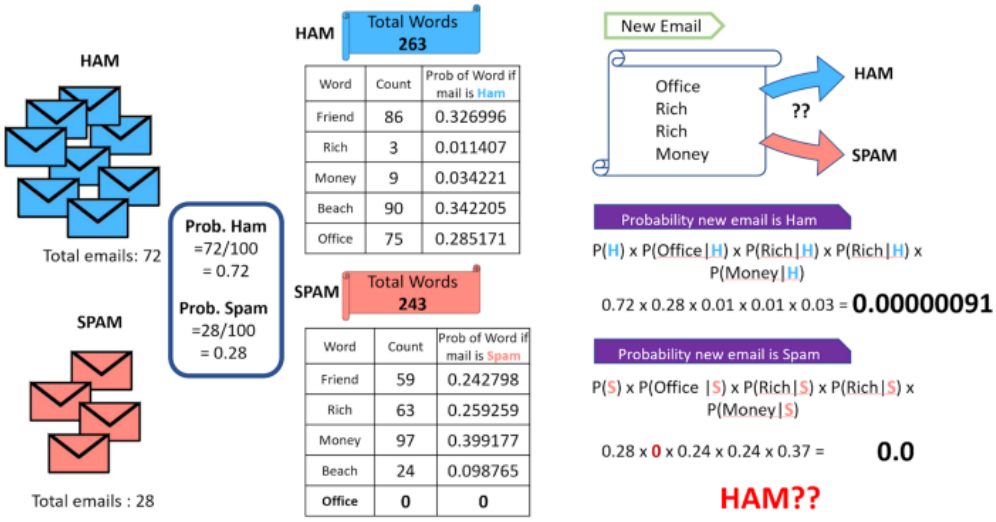
And then using the Naive Bayes, assuming that the occurrence of each word is independent of each other, we calculate the probability of a new email containing the words 'friend', 'rich', 'beach', 'money'.



The Zero frequency problem:

If an individual class label is missing, then the frequency-based probability estimate will be zero. And we will get a zero when all the probabilities are multiplied.

In our above example, if the word 'office' is missing from the list of words in spam, whenever we see a new email that consists of the word 'office' it will always be considered a ham, no matter how 'spammy' the other words are.

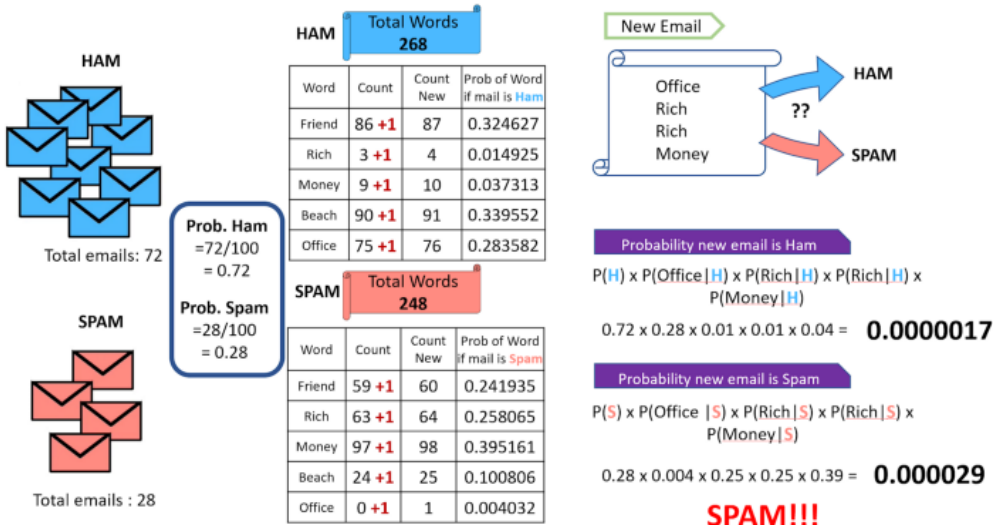


The solution to Zero Frequency Problem:

An approach to overcome this 'zero-frequency problem' is to add one to the count for every attribute value-class combination when an attribute value doesn't occur with every class value.

This will lead to the removal of all the zero values from the classes and, at the same time, will not impact the overall relative frequency of the classes.

In the example above, we increase the value of the word count by 1 for both the spam and the ham emails. Then we calculate the probability of a new email with the words 'office', 'rich', 'rich', 'money' as being spam or ham.



This process of 'smoothing' our data by adding a number is known as additive smoothing, also called Laplace smoothing.

Finding the probability for continuous valued features

Gaussian Naive Bayes Classifier

Person	Height (ft)	Weight (lbs)	Foot size (inches)
Male	6.00	180	12
Male	5.92	190	11
Male	5.58	170	12
Male	5.92	165	10
Female	5.00	100	6
Female	5.50	150	8
Female	5.42	130	7
Female	5.75	150	9

Based on the following data determine the gender of a person having height 6 ft., weight 130 lbs, and foot size 8 inch. (use Naive Bayes algorithm).

Person	Height (ft)	Weight (lbs)	Foot size (inches)
Male	6.00	180	12
Male	5.92	190	11
Male	5.58	170	12
Male	5.92	165	10
Female	5.00	100	6
Female	5.50	150	8
Female	5.42	130	7
Female	5.75	150	9

$$P(Male) = 4/8 = 0.5$$

$$P(Female) = 4/8 = 0.5$$

Male:

$$Mean\ (Height) = \frac{(6+5.92+5.58+5.92)}{4} = 5.855$$

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

$$= \frac{(6-5.855)^2+(5.92-5.855)^2+(5.58-5.855)^2+(5.92-5.855)^2}{4-1}$$

$$= 0.035055$$

Person	Height (ft)	Weight (lbs)	Foot size (inches)
Male	6.00	180	12
Male	5.92	190	11
Male	5.58	170	12
Male	5.92	165	10
Female	5.00	100	6
Female	5.50	150	8
Female	5.42	130	7
Female	5.75	150	9

$P(Male) = 4/8 = 0.5$

$P(Female) = 4/8 = 0.5$

Male:

$Mean\ (Height) = \frac{(6+5.92+5.58+5.92)}{4} = 5.855$

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

$= \frac{(6-5.855)^2+(5.92-5.855)^2+(5.58-5.855)^2+(5.92-5.855)^2}{4-1}$

$= 0.035055$

Sex	Mean (height)	Variance (height)	Mean (weight)	Variance (weight)	Mean(foot size)	Variance (foot size)
Male	5.855	0.035033	176.25	122.92	11.25	0.91667
Female	5.4175	0.097225	132.5	0558.33	7.5	1.6667

Sex	Mean (height)	Variance (height)	Mean (weight)	Variance (weight)	Mean(foot size)	Variance (foot size)
Male	5.855	0.035033	176.25	122.92	11.25	0.91667
Female	5.4175	0.097225	132.5	0558.33	7.5	1.6667

New Instance to be Classified is:

Sex	Height(ft)	Weight(lbs)	Foot size(inch)
Sample	6	130	8

$P(Male) = 4/8 = 0.5$

$P(Female) = 4/8 = 0.5$

$$Posterior\ (Male) = \frac{P(M) * P(H|M) * P(W|M) * P(FS|M)}{Evidence}$$

$$Posterior\ (Female) = \frac{P(F) * P(H|F) * P(W|F) * P(FS|F)}{Evidence}$$

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

$$P(H|M) = \frac{1}{\sqrt{2 * 3.142 * 0.035033}} * e^{-\frac{(6-5.855)^2}{2*0.035033}} = 1.5789$$

$$P(W|M) = 5.9881e^{-6}$$

$$P(FS|M) = 1.3112e^{-3}$$

$$P(H|F) = 2.2346e^{-1}$$

$$P(W|F) = 1.6789e^{-2}$$

$$P(FS|F) = 2.8669e^{-1}$$

Sex	Mean (height)	Variance (height)	Mean (weight)	Variance (weight)	Mean(foot size)	Variance (foot size)
Male	5.855	0.035033	176.25	122.92	11.25	0.91667
Female	5.4175	0.097225	132.5	0558.33	7.5	1.6667

New Instance to be Classified is:

Sex	Height(ft)	Weight(lbs)	Foot size(inch)
Sample	6	130	8

$$P(Male) = 4/8 = 0.5$$
$$P(Female) = 4/8 = 0.5$$

$$P(H|M) = \frac{1}{\sqrt{2 * 3.142 * 0.035033}} * e^{-\frac{(6-5.855)^2}{2*0.035033}} = 1.5789$$
$$P(W|M) = 5.9881e^{-6}$$
$$P(FS|M) = 1.3112e^{-3}$$

$$P(H|F) = 2.2346e^{-1}$$
$$P(W|F) = 1.6789e^{-2}$$
$$P(FS|F) = 2.8669e^{-1}$$

Female

$$Posterior\ (Male) = \frac{P(M)*P(H|M)*P(W|M)*P(FS|M)}{Evidence} = 0.5 * 1.5789 * 5.9881e^{-6} * 1.3112e^{-3} = 6.1984e^{-9}$$

$$Posterior\ (Female) = \frac{P(F)*P(H|F)*P(W|F)*P(FS|F)}{Evidence} = 0.5 * 2.2346e^{-1} * 1.6789e^{-2} * 2.8669e^{-1} = 5.377e^{-4}$$

Text Classification using Naive Bayes.

Business Case: Sentiment Analysis

- ABC Restaurant intends to build a binary classification model (positive/ negative) for customer reviews received on their facebook page
- Business intends to build an inhouse customer support team to call-back customers who gave negative feedbacks, and resolve their issues/complaints, ensuring they revisit
- Restaurant has shared following datasets:
  - Historical reviews - along with labels
  - Fresh reviews - without labels
    - Client wants us to generate labels for this

Customer Reviews	Review Label
Review 1	Positive
Review 2	Negative
Review 3	Positive

- **Historical review dataset** - with positive/negative (1/0) labels for model preparation - 900 observations

Review	Liked
Wow... Loved this place.	1
Crust is not good.	0
Not tasty and the texture was just nasty.	0
Stopped by during the late May bank holiday off Rick Steve recommendation and loved it.	1
The selection on the menu was great and so were the prices.	1

- **Current week's review dataset** - for which labels are to be generated - 100 observations

Review	Liked
I'm super pissd.	
And service was super friendly.	
Why are these sad little vegetables so overcooked?	
This place was such a nice surprise!	
They were golden-crispy and delicious.	

Original Reviews-

Intuition - Data Cleaning

1. Wow... Loved this place.
2. Crust is not good.
3. Not tasty and the texture was just nasty.
4. Also there are combos like a burger, fries, and beer for 23 which is a decent deal.
5. I would definitely recommend the wings as well as the pizza.
6. Highly recommended.

Dropping Speical Characters and #'s

1. Wow... Loved this place.
2. Crust is not good.
3. Not tasty and the texture was just nasty.
4. Also there are combos like a burger, fries, and beer for 23 which is a decent deal.
5. I would definitely recommend the wings as well as the pizza.
6. Highly recommended.

1. Wow Loved this place
2. Crust is not good
3. Not tasty and the texture was just nasty
4. Also there are combos like a burger fries and beer for which is a decent deal
5. I would definitely recommend the wings as well as the pizza
6. Highly recommended

Converting to small-case

1. Wow Loved this place
2. Crust is not good
3. Not tasty and the texture was just nasty
4. Also there are combos like a burger fries and beer for which is a decent deal
5. I would definitely recommend the wings as well as the pizza
6. Highly recommended

Dropping Stopwords and Stemming

1. wow loved this place
2. crust is not good
3. not tasty and the texture was just nasty
4. also there are combos like a burger fries and beer for which is a decent deal
5. i would definitely recommend the wings as well as the pizza
6. highly recommended

1. wow loved this place
2. crust is not good
3. not tasty and the texture was just nasty
4. also there are combos like a burger fries and beer for which is a decent deal
5. i would definitely recommend the wings as well as the pizza
6. highly recommended

Stop words
To-be-stemmed words

1. wow love place
2. crust not good
3. not tasti textur nasti
4. also combo like burger fri beer decent deal
5. would definit recommend wing well pizza
6. highli recommend

Bag of Words

1. wow love place
2. crust not good
3. not tasti textur nasti
4. also combo like burger fri beer decent deal
5. would definit recommend wing well pizza
6. highli recommend

Review	Liked
wow love place	1
crust not good	0
not tasti textur nasti	0

Review	Liked
wow love place	1
crust not good	0
not tasti textur nasti	0

1. Bag of words representation discards information on order and sequencing of words
2. Dropping tokens (unique words) that only reflect in only a few reviews, reduces sparsity

wow	love	place	crust	not	good	tasti	textur	nasti	Liked
1	1	1	0	0	0	0	0	0	1
0	0	0	1	1	1	0	0	0	0
0	0	0	0	1	0	1	1	1	0

Naive Bayes Algorithm

```
1 import numpy as np
2 import pandas as pd
```

```
1 df = pd.read_csv('User_Data.csv')
2 df
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

```
1 x = df.iloc[:, 2:4].values
2 y = df.iloc[:, -1].values
```

```
1 from sklearn.model_selection import train_test_split
2 x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42, stratify = y)
```

```
1 from sklearn.preprocessing import StandardScaler
2 sc = StandardScaler()
3 x_train = sc.fit_transform(x_train)
4 x_test = sc.transform(x_test)
```

```
1 from sklearn.naive_bayes import GaussianNB
2 nb = GaussianNB()
3 nb.fit(x_train, y_train)
```

```
GaussianNB()
```

```
1 from sklearn.metrics import confusion_matrix, accuracy_score
2 y_pred = nb.predict(x_test)
3 cm = confusion_matrix(y_test, y_pred)
4 print(cm)
5 print(accuracy_score(y_test, y_pred))
```

```
[[47  4]
 [ 6 23]]
0.875
```

```
1
```

Internal Working of Naive Bayes

```
1 df2 = pd.read_csv('play_tennis.csv')
2 df2
```

	day	outlook	temp	humidity	wind	play
0	D1	Sunny	Hot	High	Weak	No
1	D2	Sunny	Hot	High	Strong	No
2	D3	Overcast	Hot	High	Weak	Yes
3	D4	Rain	Mild	High	Weak	Yes
4	D5	Rain	Cool	Normal	Weak	Yes
5	D6	Rain	Cool	Normal	Strong	No
6	D7	Overcast	Cool	Normal	Strong	Yes
7	D8	Sunny	Mild	High	Weak	No
8	D9	Sunny	Cool	Normal	Weak	Yes
9	D10	Rain	Mild	Normal	Weak	Yes

```
1 df2['play'].value_counts()
```

```
Yes    9
No     5
Name: play, dtype: int64
```

```
1 pyes = 9/14
2 pno = 5/14
```

```
1 pd.crosstab(df2['outlook'], df2['play'])
```

play	No	Yes
outlook		
Overcast	0	4
Rain	2	3
Sunny	3	2

```
1 pon = 0
2 prn = 2/5
3 psn = 3/5
4
5 psy = 4/9
6 pry = 3/9
7 psy = 2/9
```

```
1 pd.crosstab(df2['temp'], df2['play'])
```

play	No	Yes
temp		
Cool	1	3
Hot	2	2
Mild	2	4

```
1 pcn = 1/5
2 phn = 2/5
3 pmn = 2/5
4
5 pcy = 3/9
6 phy = 2/9
7 pmy = 4/9
```

```
1 pd.crosstab(df2['humidity'], df2['play'])
```

play	No	Yes
humidity		
High	4	3
Normal	1	6

```
# No
phighn = 4/5
pnormaln = 1/5

# Yes
phighy = 3/9
pnormaly = 6/9

```

```
pd.crosstab(df2['wind'], df2['play'])
```

play	No	Yes
wind		
Strong	3	3
Weak	2	6

```
# No
pstrongn = 3/5
pweakn = 2/5

# Yes
pstrongy = 3/9
pweaky = 6/9

```

```
# Now find the probability of playing tennis by the formaula from above

# outlook=Sunny ,Temp=Hot ,Humididty=High ,Wind=Weak

# find for this above

pyes = pyes * phy * phighy * pweaky
pyes
```

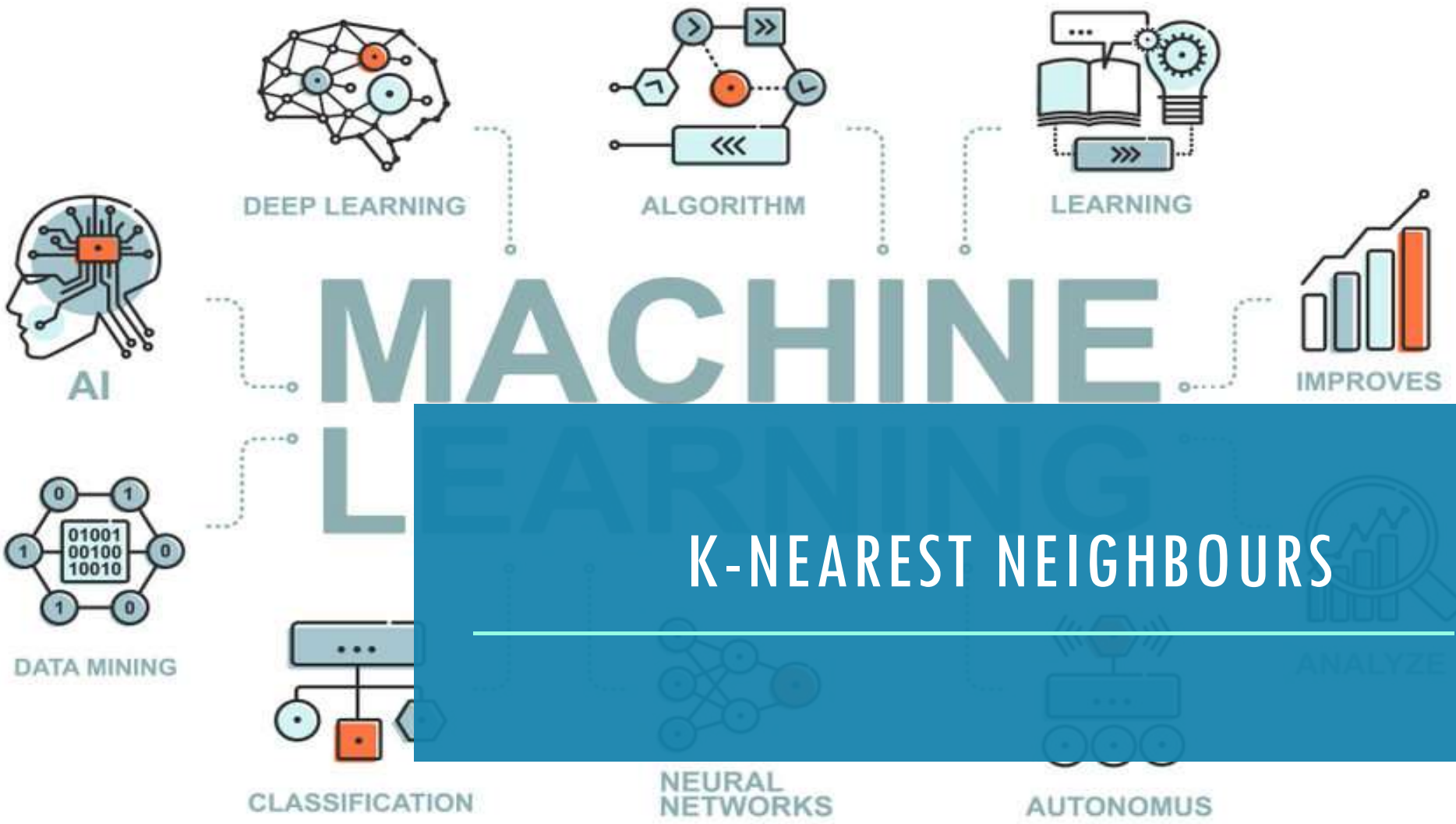
```
0.031746031746031744
```

```
# No
pno = pno * phn * phighn * pweakn
pno
```

```
0.04571428571428573
```

```

```



K-NEAREST NEIGHBOURS

confusion matrices

In [18]:

```
grid_predictions = grid.predict(X_test)
print(confusion_matrix(y_test,grid_predictions))
print(classification_report(y_test,grid_predictions))
```

```
[[12  0  0]
 [ 0  9  3]
 [ 0  0  6]]
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        12
Iris-versicolor       1.00      0.75      0.86        12
 Iris-virginica       0.67      1.00      0.80         6

       accuracy                           0.90        30
      macro avg       0.89      0.92      0.89        30
   weighted avg       0.93      0.90      0.90        30
```

In [ ]:

```
```

K-Nearest Neighbor(KNN) Algorithm for Machine Learning

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- 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.
- 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.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.
- 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.
- 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:

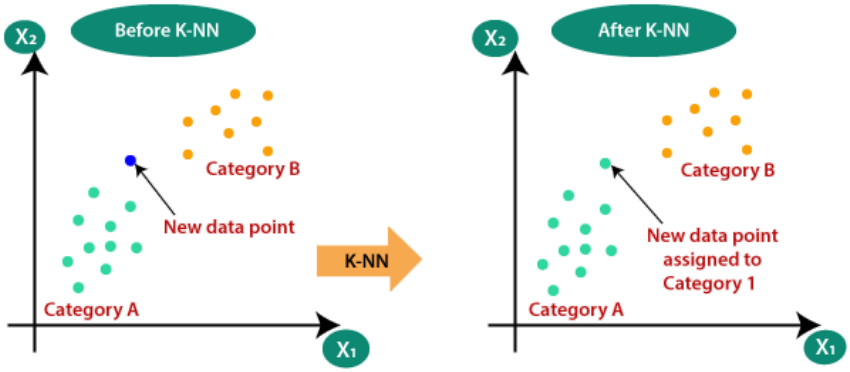
Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog.

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 neighbors

Step-2:

Calculate the Euclidean distance of K number of neighbors

Step-3:

Take the K nearest neighbors as per the calculated Euclidean distance.

Step-4:

Among these k neighbors, count the number of the data points in each category.

Step-5:

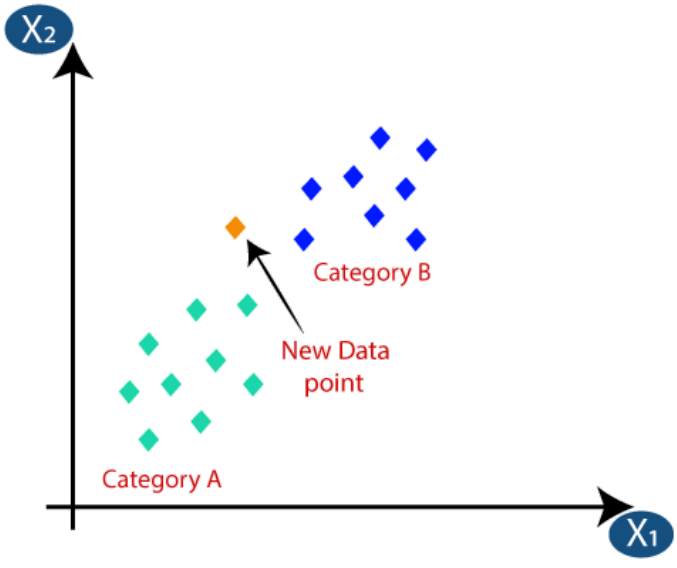
Assign the new data points to that category for which the number of the neighbor is maximum.

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:



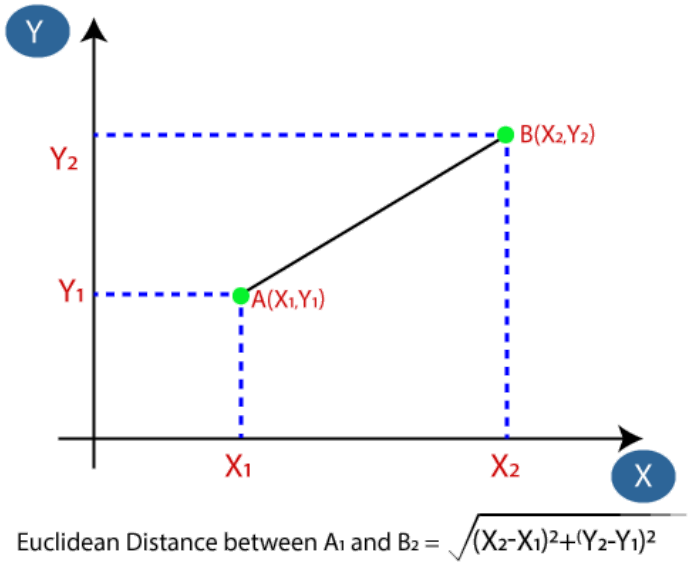
Firstly,

- we will choose the number of neighbors, so we will choose the k=5.

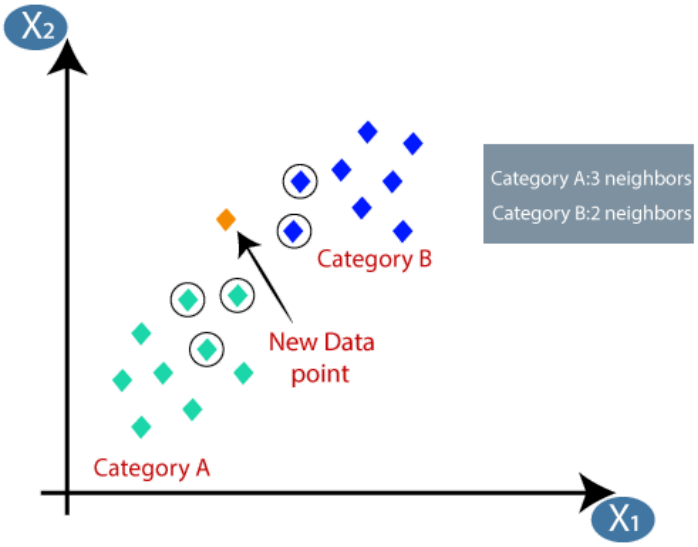
Next,

- 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:



- By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:



As we can see the 3 nearest neighbors 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:

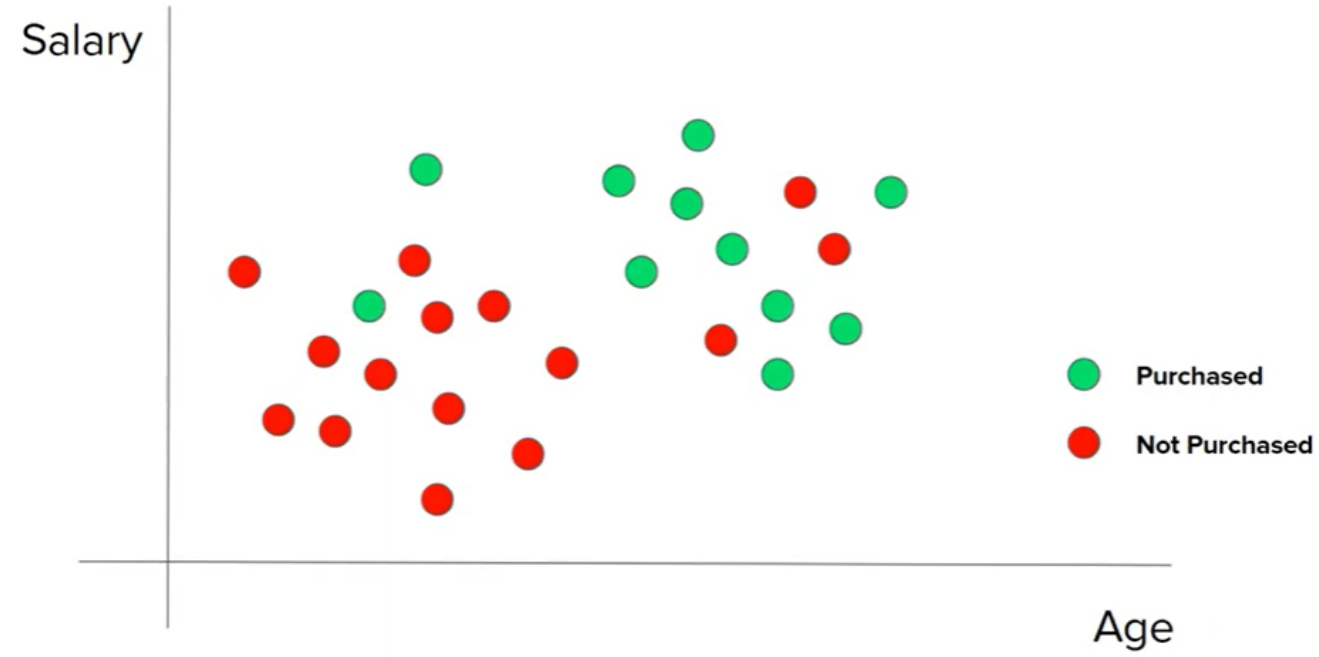
- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.

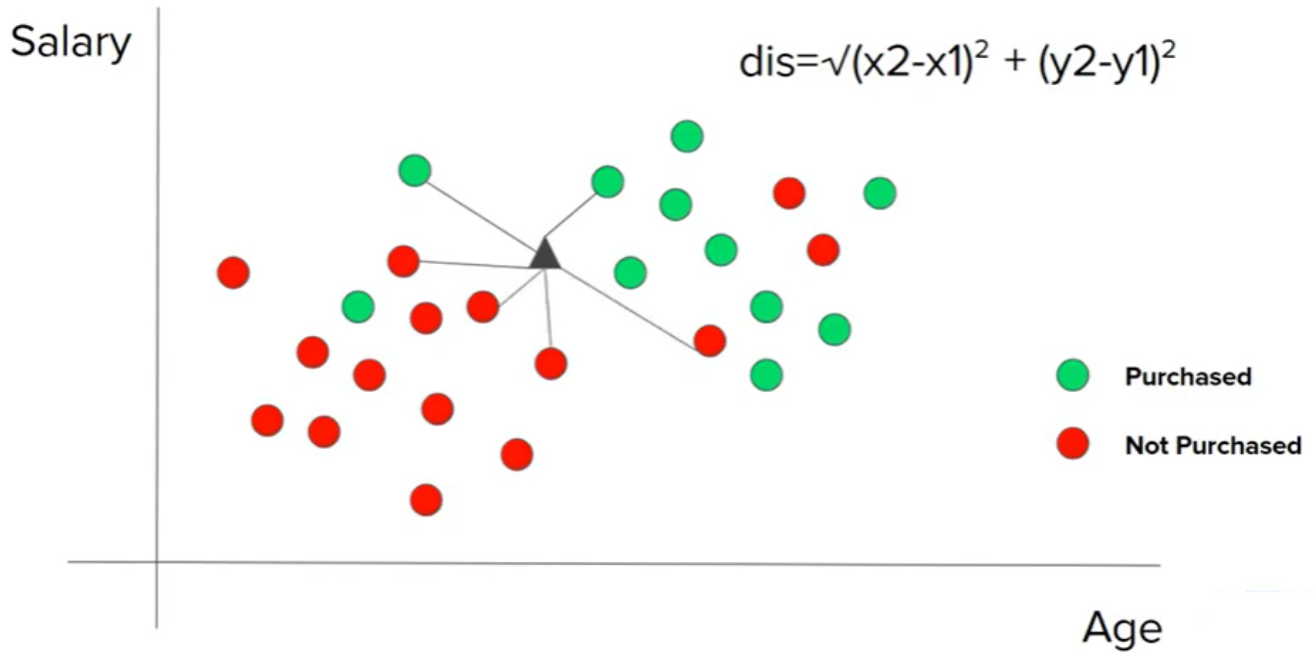
Assumptions

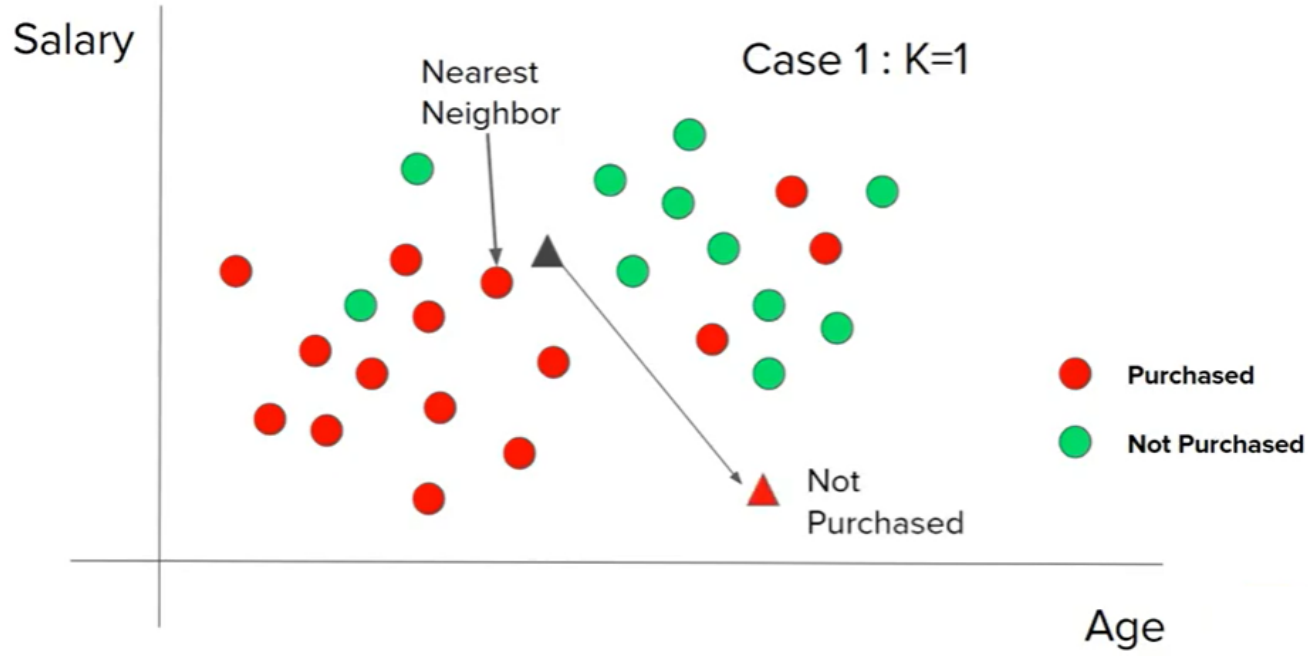
- Knn assumes data is in metric space and there is a notion of distance
- Each of the training data consists of a label data associated with it, either + or -. Although Knn also supports multiclass classification.
- We are also given a single number "k" . This number decides how many neighbors influence the classification. This is usually a odd number

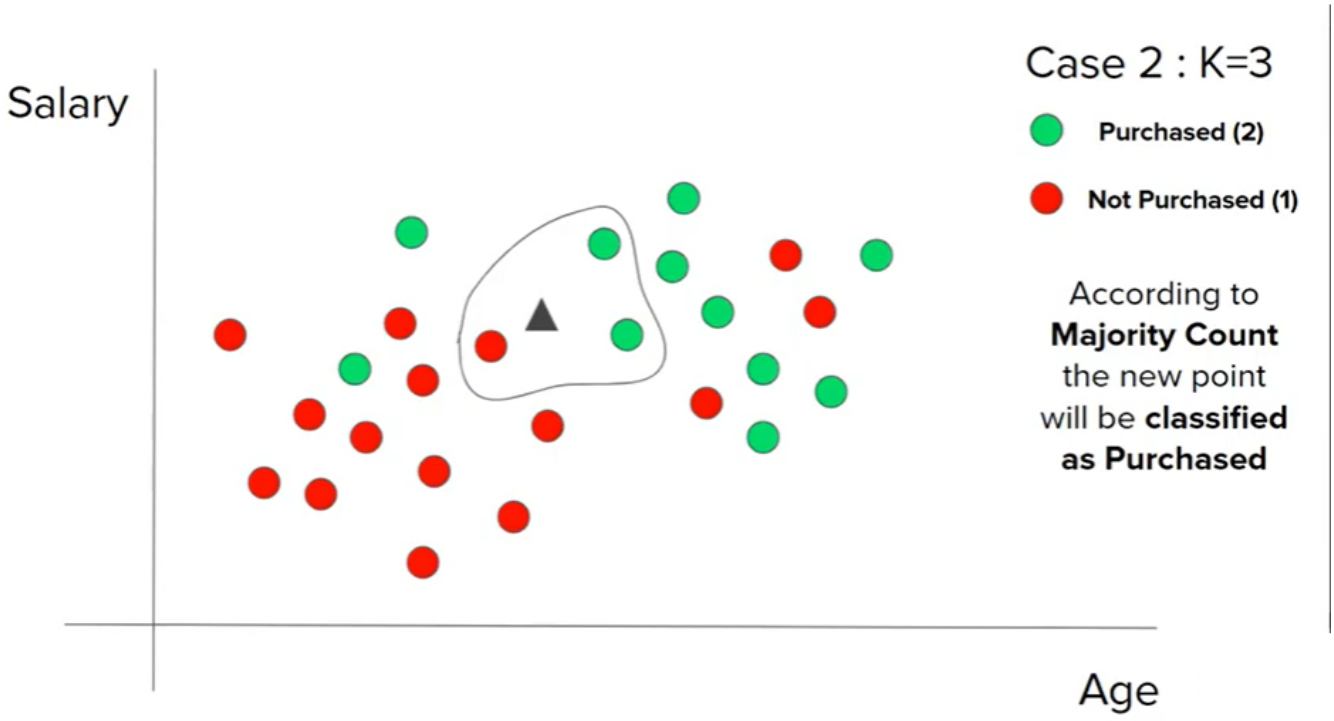
Geometric Intuition

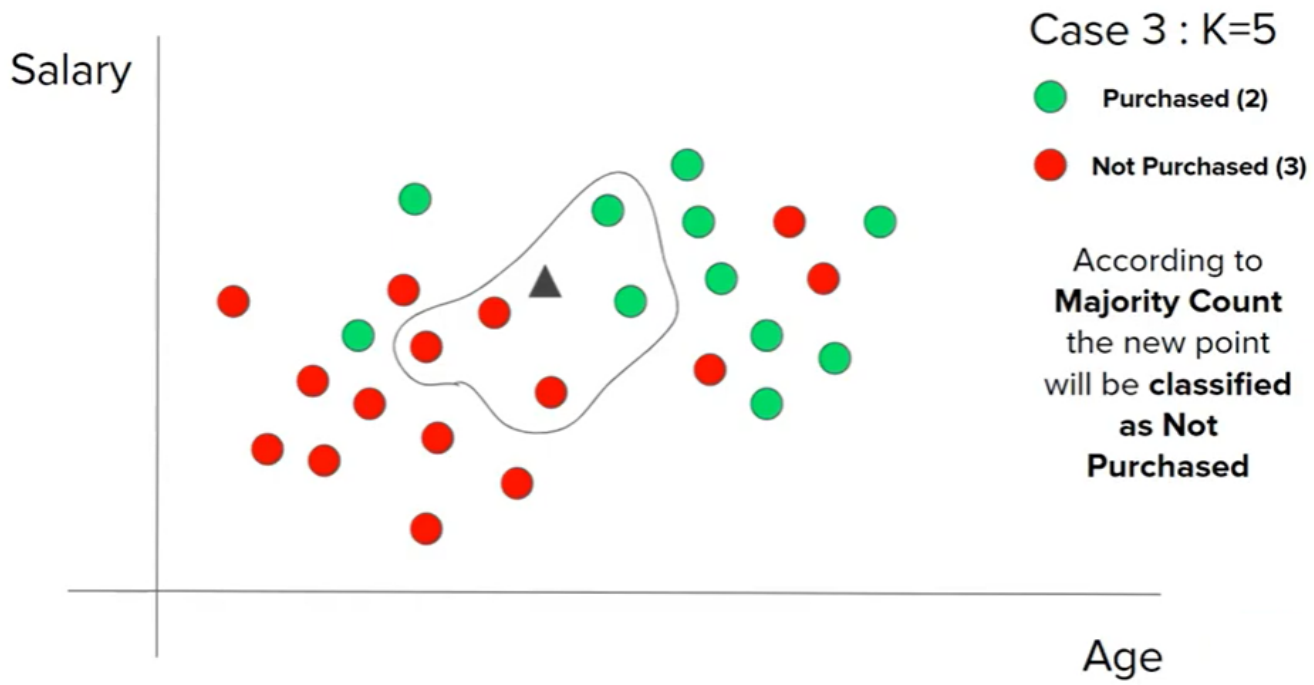
S. No	Age	Salary(in Thousands)	Purchased
1	25	20	N
2	63	120	Y
3	33	75	N
4	42	100	Y
...	...	...	...


Salary
Purchased
Not Purchased
Age

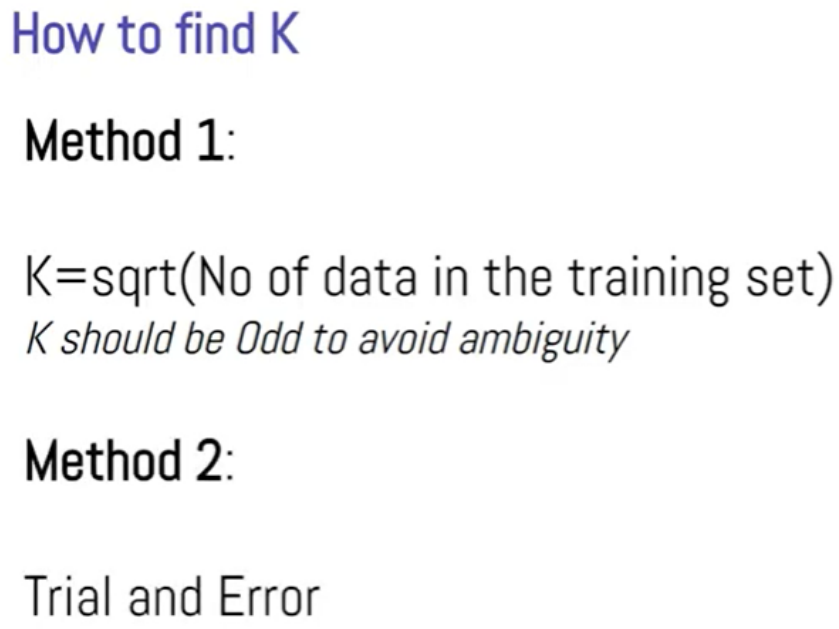

Salary
dis=√(x2-x1)² + (y2-y1)²
Purchased
Not Purchased
Age


Salary
Nearest
Neighbor
Case 1 : K=1
Purchased
Not Purchased
Not
Purchased
Age


Salary
Case 2 : K=3
Purchased (2)
Not Purchased (1)
According to
Majority Count
the new point
will be classified
as Purchased
Age



How to Find K



In [ ]:

```
# python code
```

In [14]:

```
import numpy as np
import pandas as pd
```

In [15]:

```
data=pd.read_csv("Social_Network_Ads.csv")
```

In [16]:

```
data.head()
```

Out[16]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

In [17]:

```
X=data.iloc[:,2:4].values
X.shape
```

Out[17]:

```
(400, 2)
```

In [18]:

```
y=data.iloc[:,-1].values
y.shape
```

Out[18]:

```
(400,)
```

In [19]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.20)
```

In [20]:

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
```

In [22]:

```
X_train=scaler.fit_transform(X_train)
X_test=scaler.transform(X_test)
```

In [ ]:

```
# K value using 1st method
```

In [23]:

```
X_train.shape
```

Out[23]:

```
(320, 2)
```

In [24]:

```
np.sqrt(X_train.shape[0])
```

Out[24]:

```
17.88854381999832
```

In [25]:

```
k=17
```

In [26]:

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=k)
```

In [27]:

```
# Train the model
knn.fit(X_train,y_train)
```

Out[27]:

```
KNeighborsClassifier(n_neighbors=17)
```

In [28]:

```
y_pred=knn.predict(X_test)
```

In [29]:

```
y_pred
```

Out[29]:

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

In [30]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
```

In [31]:

```
accuracy_score(y_test,y_pred)
```

Out[31]:

```
0.8875
```

In [33]:

```
len(y_pred)
```

Out[33]:

```
80
```

In [32]:

```
confusion_matrix(y_test,y_pred)
```

Out[32]:

```
array([[43,  5],
       [ 4, 28]], dtype=int64)
```

In [24]:

```
# method 2 Trial and Error Method
accuracy1=[]
for i in range(1,26):
    knn=KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    accuracy1.append(accuracy_score(y_test,knn.predict(X_test)))

sorted(accuracy1,reverse=True)
```

Out[24]:

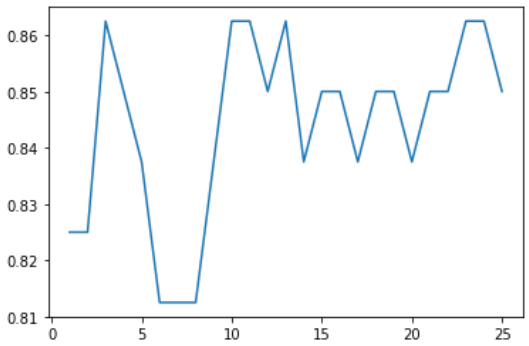
```
[0.8625,
 0.8625,
 0.8625,
 0.8625,
 0.8625,
 0.8625,
 0.85,
 0.85,
 0.85,
 0.85,
 0.85,
 0.85,
 0.85,
 0.85,
 0.85,
 0.8375,
 0.8375,
 0.8375,
 0.8375,
 0.8375,
 0.825,
 0.825,
 0.8125,
 0.8125,
 0.8125]
```

In [25]:

```
import matplotlib.pyplot as plt
plt.plot(range(1,26),accuracy1)
```

Out[25]:

```
[<matplotlib.lines.Line2D at 0x1c5d9edb8b0>]
```



In [28]:

```
knn=KNeighborsClassifier(n_neighbors=11)
knn.fit(X_train,y_train)
y_pred=knn.predict(X_test)
accuracy_score(y_test,y_pred)
```

Out[28]:

```
0.8625
```

Python implementation of the KNN algorithm

To do the Python implementation of the K-NN algorithm, we will use the same problem and dataset which we have used in Logistic Regression. But here we will improve the performance of the model.

Below is the problem description:

Mb>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.

Below is the dataset:

User ID	Gender	Age	EstimatedSalary	Purchased
15624510	Male	19	19000	0
15810944	Male	35	20000	0
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	0
15728773	Male	27	58000	0
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	0
15727311	Female	35	65000	0
15570769	Female	26	80000	0
15606274	Female	26	52000	0
15746139	Male	20	86000	0
15704987	Male	32	18000	0
15628972	Male	18	82000	0
15697686	Male	29	80000	0
15733883	Male	47	25000	1
15617482	Male	45	26000	1
15704583	Male	46	28000	1
15621083	Female	48	29000	1
15649487	Male	45	22000	1
15736760	Female	47	49000	1

Steps to implement the K-NN algorithm:

Data Pre-processing step

Fitting the K-NN algorithm to the Training set

Predicting the test result

Test accuracy of the result(Creation of Confusion matrix)

Visualizing the test set result.

Data Pre-Processing Step:

The Data Pre-processing step will remain exactly the same as Logistic Regression. Below is the code for it:

In [1]:

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

In [2]:

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

Out[2]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

In [3]:

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

In [4]:

```
# 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)
```

Fitting K-NN classifier to the Training data:

Now we will fit the K-NN classifier to the training data. To do this we will import the KNeighborsClassifier class of Sklearn Neighbors library. After importing the class, we will create the Classifier object of the class. The Parameter of this class will be

n_neighbors:

To define the required neighbors of the algorithm. Usually, it takes 5.

In [11]:

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

Out[11]:

```
KNeighborsClassifier()
```

Predicting the Test Result:

To predict the test set result, we will create a y_pred vector as we did in Logistic Regression.

Below is the code for it:

In [12]:

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

Creating the Confusion Matrix:

Now we will create the Confusion Matrix for our K-NN model to see the accuracy of the classifier. Below is the code for it:

In [13]:

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

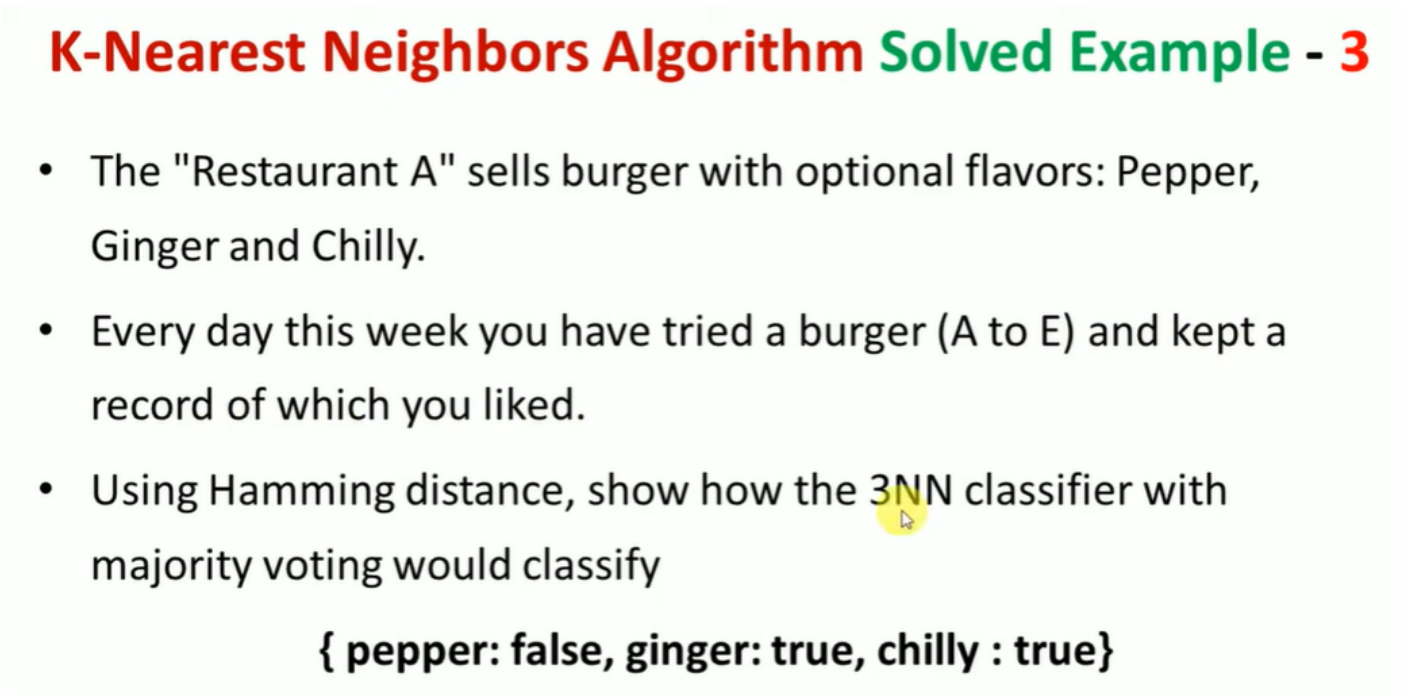
Out[13]:

```
array([[59,  9],
       [ 8, 24]], dtype=int64)
```

In the above image, we can see there are 59+24= 83 correct predictions and 8+9= 17 incorrect predictions,

Handling Categorical Data

	Pepper	Ginger	Chilly	Liked
A	True	True	True	False
B	True	False	False	True
C	False	True	True	False
D	False	True	False	True
E	True	False	False	True



	Pepper	Ginger	Chilly	Liked
A	True	True	True	False
B	True	False	False	True
C	False	True	True	False
D	False	True	False	True
E	True	False	False	True

New Example - Q: pepper: false, ginger: true, chilly : true

- But How to calculate the distance for attributes with nominal or categorical values.
- Here we can use Hamming distance to find the distance between the categorical values.
- Let x1 and x2 are the attribute values of two instances.
- Then, in hamming distance, if the categorical values are same or matching that is x1 is same as x2 then distance is 0, otherwise 1.
- For example,
- If value of **x1 is blue** and **x2 is also blue** then the distance between x1 and x2 is **0**.
- If value of **x1 is blue** and **x2 is red** then the distance between x1 and x2 is **1**.

	Pepper	Ginger	Chilly	Liked	Distance
A	True	True	True	False	1 + 0 + 0 = 1
B	True	False	False	True	1 + 1 + 1 = 3
C	False	True	True	False	0 + 0 + 0 = 0
D	False	True	False	True	0 + 0 + 1 = 1
E	True	False	False	True	1 + 1 + 1 = 3

New Example - Q: pepper: false, ginger: true, chilly : true

Use Hamming Distance and

find the distance from Query Example (Q) to training examples (A-E)

	Pepper	Ginger	Chilly	Liked	Distance
A	True	True	True	False	1 + 0 + 0 = 1
B	True	False	False	True	1 + 1 + 1 = 3
C	False	True	True	False	0 + 0 + 0 = 0
D	False	True	False	True	0 + 0 + 1 = 1
E	True	False	False	True	1 + 1 + 1 = 3

New Example - Q: pepper: false, ginger: true, chilly : true

New Sample will be classified as False

Cross Validation

Cross-Validation is used for evaluate predictive models by partitioning the original sample into a training set to train the model, and a test set to evaluate it.

Cross-Validation in Sklearn is very helpful for us to select the correct Model and Model parameters. By using that, we can intuitively see the effect of different Models or parameters on the structural accuracy.

We are going to use the famous dataset 'iris' with the KNN Classifier

1. Use knn.score() to see the accuracy

Basically, the accuracy from knn.score() only test one set of train and test dataset.

In [14]:

```
# We are going to use the famous dataset 'iris' with the KNN Classifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
```

In [15]:

```
# load dataset
iris = load_iris()
X = iris.data
y = iris.target
```

In [16]:

```
# split into test and train dataset, and use random_state=48
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=4)
```

In [17]:

```
# build KNN model and choose n_neighbors = 5
knn = KNeighborsClassifier(n_neighbors = 5)
```

In [18]:

```
# train the model
knn.fit(X_train, y_train)
```

Out[18]:

```
KNeighborsClassifier()
```

In [19]:

```
# get the predict value from X_test
y_pred = knn.predict(X_test)
```

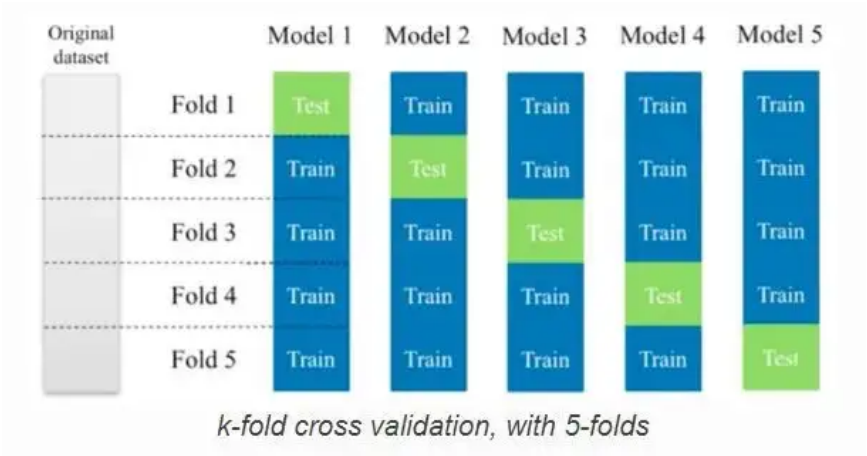
In [20]:

```
# print the score
print('accuracy: ', knn.score(X_test, y_test))
# accuracy:  0.973684210526
```

```
accuracy:  0.9736842105263158
```

2. Cross-Validation for Classification

In k-fold cross-validation, the original sample is randomly partitioned into k equal size subsamples.



In [21]:

```
# import k-folder
from sklearn.model_selection import cross_val_score
```

In [22]:

```
# use the same model as before
knn = KNeighborsClassifier(n_neighbors = 5)
# X,y will automatically devided by 5 folder, the scoring I will still use the accuracy
scores = cross_val_score(knn, X, y, cv=5, scoring='accuracy')
# print all 5 times scores
print(scores)
# [ 0.96666667  1.          0.93333333  0.96666667  1.        ]
# then I will do the average about these five scores to get more accuracy score.
print(scores.mean())
# 0.973333333333
```

```
[0.96666667 1.         0.93333333 0.96666667 1.        ]
0.9733333333333334
```

Curse of Dimensionality

K-Nearest Neighbor(KNN) Algorithm

Program for KNN

```
1 import numpy as nm
2 import matplotlib.pyplot as mtp
3 import pandas as pd
```

```
1 data_set= pd.read_csv('User_Data.csv')
2 data_set
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

```
1 x= data_set.iloc[:, [2,3]].values
2 y= data_set.iloc[:, 4].values
```

```
1 from sklearn.model_selection import train_test_split
2 x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
```

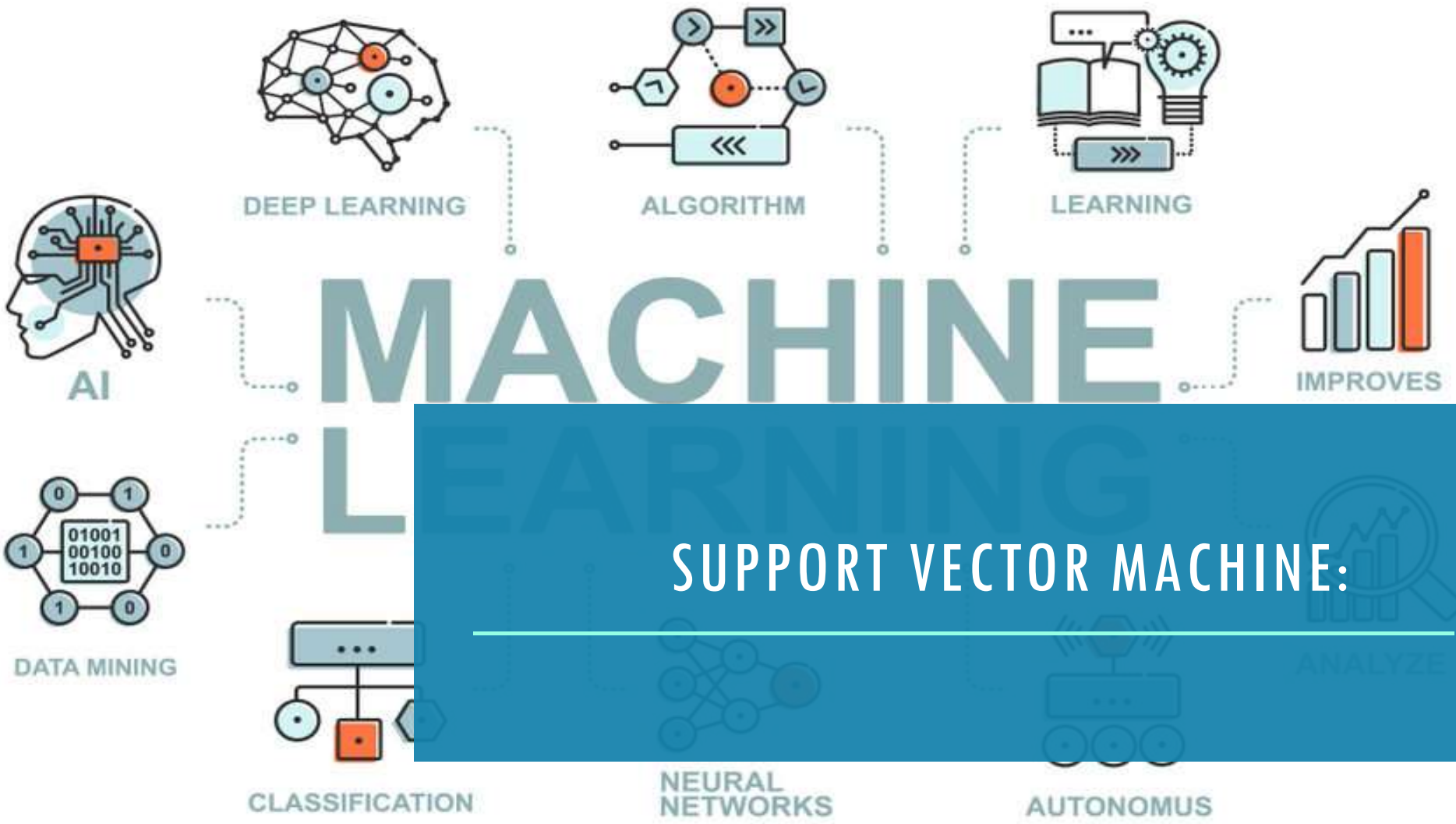
```
1 from sklearn.neighbors import KNeighborsClassifier
2 classifier= KNeighborsClassifier(n_neighbors=15, metric='minkowski', p=2 )
3 classifier.fit(x_train, y_train)
```

```
1 y_pred= classifier.predict(x_test)
```

```
1 from sklearn.metrics import confusion_matrix, accuracy_score
2 cm= confusion_matrix(y_test, y_pred)
3 print(accuracy_score(y_test, y_pred))
4 cm
```

```
0.86
array([[64,  4],
       [10, 22]])
```

We can see above that there are 64+22= 86 correct predictions and 10+4= 14 incorrect predictions



SUPPORT VECTOR MACHINE:

Support Vector Machine

17 December 2022 12:48 PM

→ Supervised Machine Learning Algo

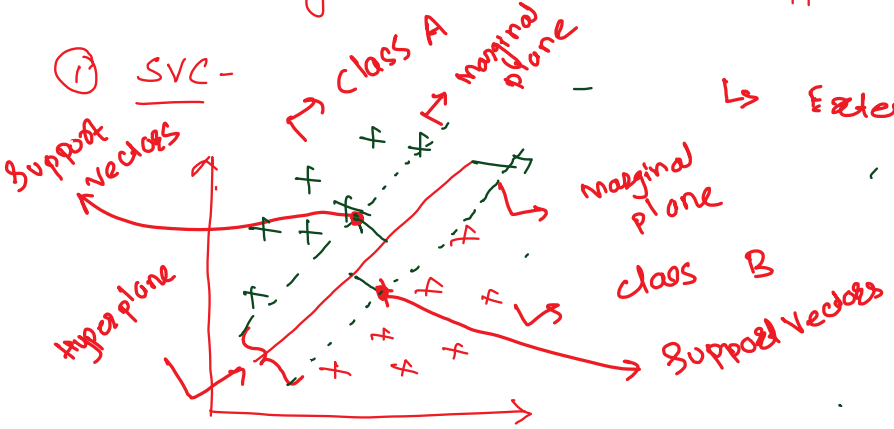
→ It uses for

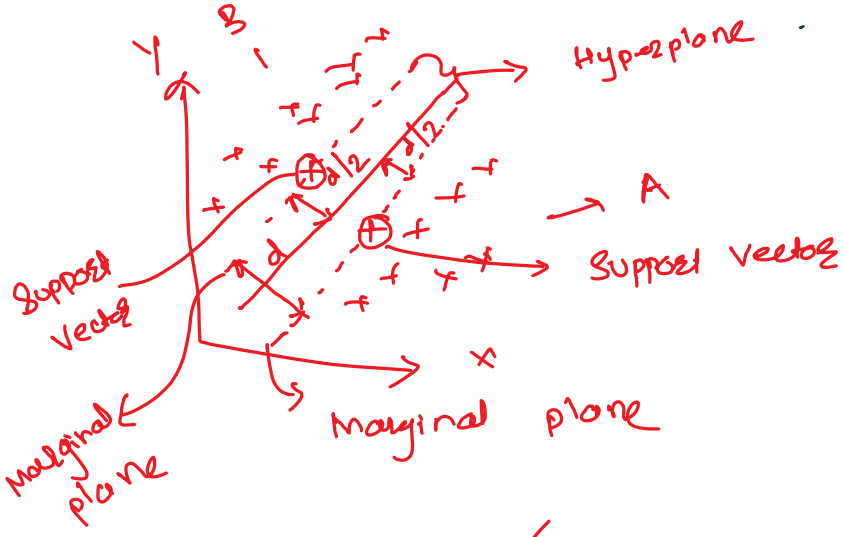
↳ classification (SVC) – Support Vector Classifier

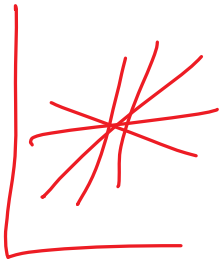
↳ Regression (SVR) – Support Vector Regression

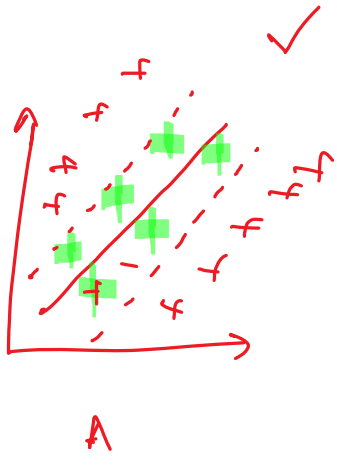
① SVC –

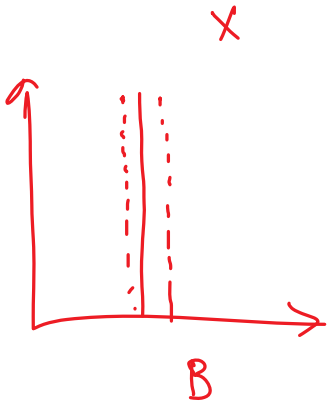
↳ Extension of logistic Regression









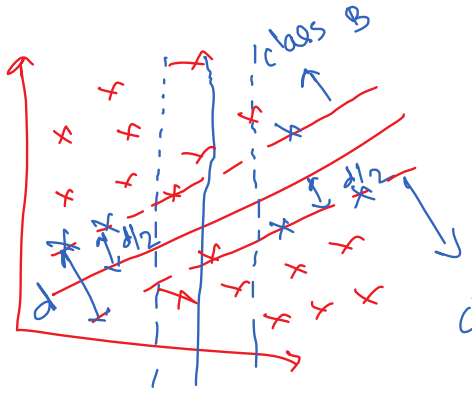


→ fig A is having Maxm size in hyperplane

→ Aim:

Distance Need to be Maximized in Hyperplane

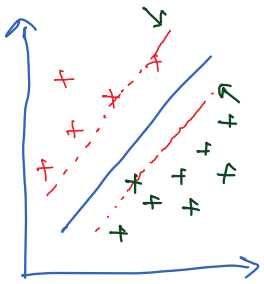
Distance Need to be maximized in hyperplane



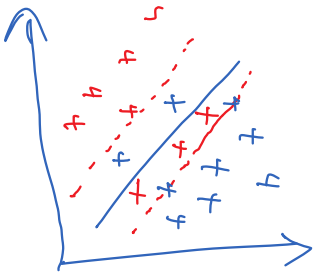
" Need to find out hyperplane with max^m distance "

* Types of Marginal plane : →

1. Hard Marginal plane
2. Soft Marginal plane



1. Soft Marginal plane



2. Hard Marginal plane

How to achieve goal ?

$y = mx + c$ (S.L.R)

$y = \theta_0 + \theta_1 x$

$y = \beta_0 + \beta_1 x$

↳ $y = w_1 x_1 + b$ → (S.L.R.)

$y = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$ (M.L.R)

↳ $\boxed{y = w^T x + b}$ → eqn

form $\begin{cases} ax + by + c = 0 \rightarrow \text{eqn of straight line} \\ y = mx + c \end{cases}$

Same:

$ax + by + c = 0$

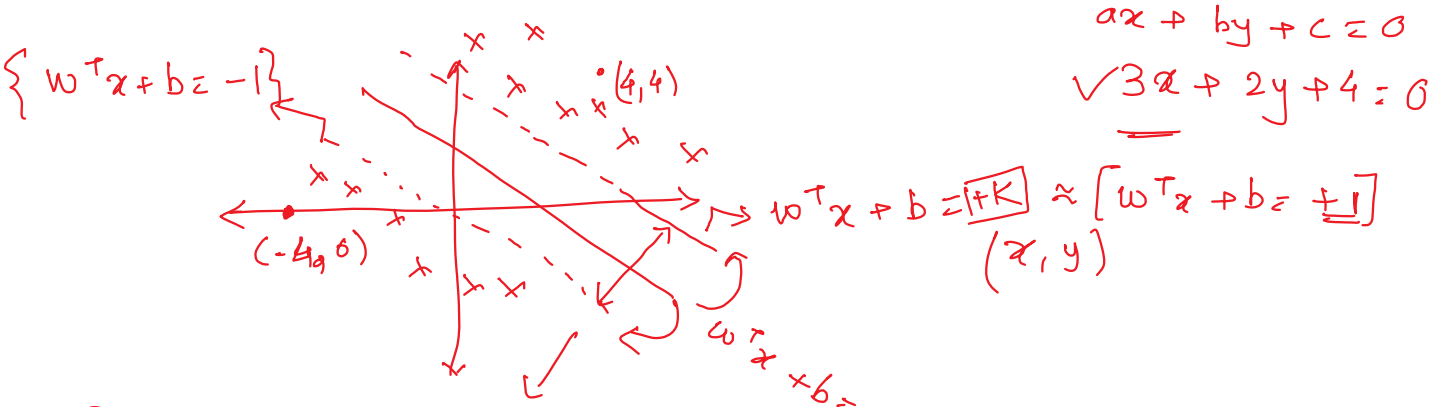
$by = -ax - c$

$\left[y = -\frac{a}{b}(x) - \frac{c}{b} \right]$

where $\left\{ m = -\frac{a}{b}, \quad c = -\frac{c}{b} \right\}$

$ax + by + c = 0$

$\sqrt{3x + 2y + 4 = 0}$

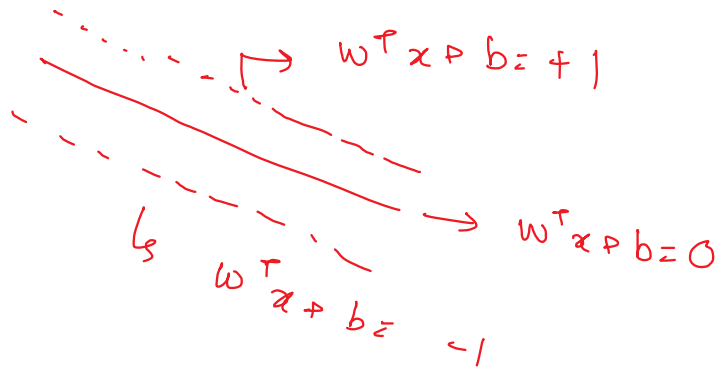


① $x = -4, \quad y = 0$

$3 \times (-4) + 2 \times (0) + 4 = -12 + 0 + 4 = -8$ -ve

② $x = 4, \quad y = 4$

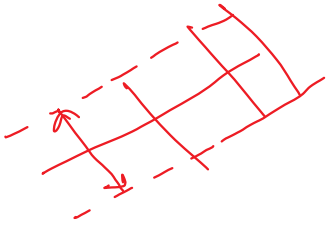
$3 \times 4 + 2 \times 4 + 4 = 12 + 8 + 4 = 24$ +ve



$$w^T x_1 + b = +1$$

$$w^T x_2 + b = -1$$

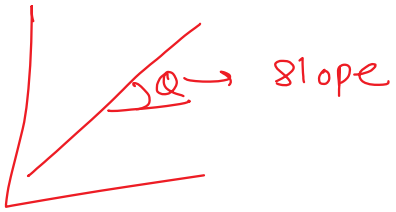
$$w^T (x_1 - x_2) = 2$$



Here $w \rightarrow$ slope

① magnitude

② Vector



$$w^T (x_1 - x_2) = 2$$

$$\frac{w^T}{\|w\|}(x_1 - x_2) = \frac{2}{\|w\|}$$ Maximize / Distance is high

finially Aim is to

$$\left\{ \begin{array}{l} \text{Maximize} \\ (w, b) \end{array} = \frac{2}{\|w\|} \right\} \rightarrow \text{Marginal plane Distance}$$

constrains such that

$$y_i = \begin{cases} +1 & w^T x + b \geq 1 \\ -1 & w^T x + b \leq -1 \end{cases}$$

* for all accurate data points

$$\{ y_i * w^T x_i + b \geq 1 \}$$

$$\text{Maximize}_{(w,b)} = \frac{2}{\|w\|} \rightarrow \text{Distance max}^m$$

↳ Maximize (w, b) = $\frac{2}{\|w\|}$ → distance more

Minimize (w, b) = $\frac{\|w\|}{2}$ → {loss}

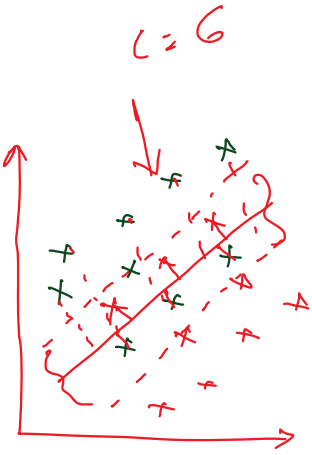
{eta - η}

↳ Cost fn: ✓

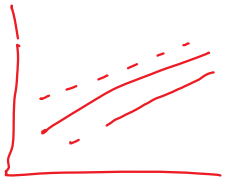
{Cost fn = minimize (w, b) = $\frac{\|w\|}{2} + C_i * \sum_{i=1}^{n} \eta_i$}

C_i : {How many points we can avoid due to Misclassification}

C = 6

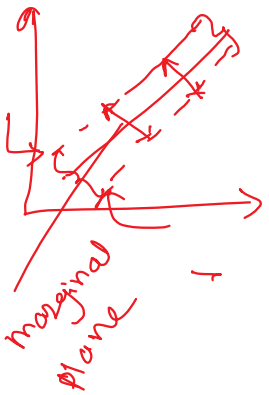


eta / η = Summation of distance of Misclassified points from marginal plane



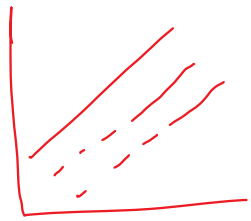
* Support Vector Regressor: →

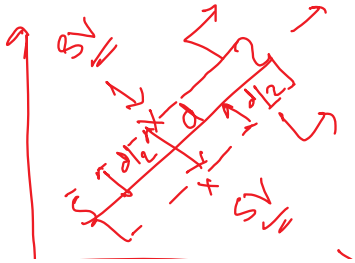
SVC

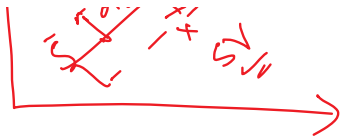


cost = minimize (w, b) $\frac{\|w\|}{2} + C \sum_{i=1}^{n} \eta_i$

Hyperparameters

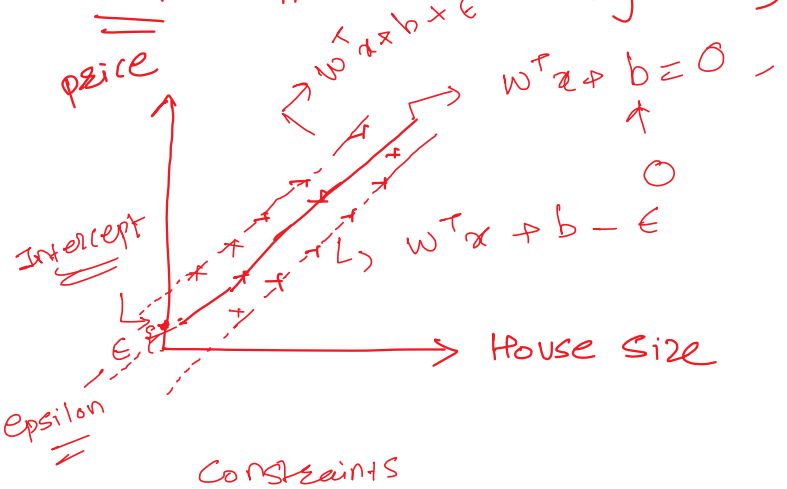


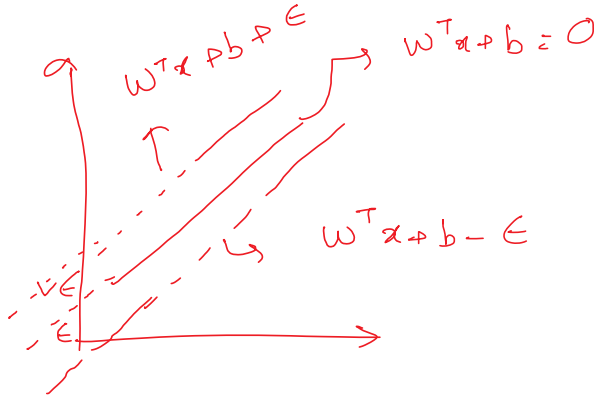




SVR (Support Vector Regressor)

price





Constraints

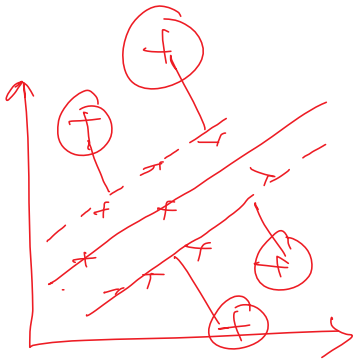
$$|y - w^T x_i| \leq \epsilon$$

MSE / MAE

↳ $|y - \hat{y}| \leq \epsilon$ → It is very good

Modified cost fⁿ eqⁿ

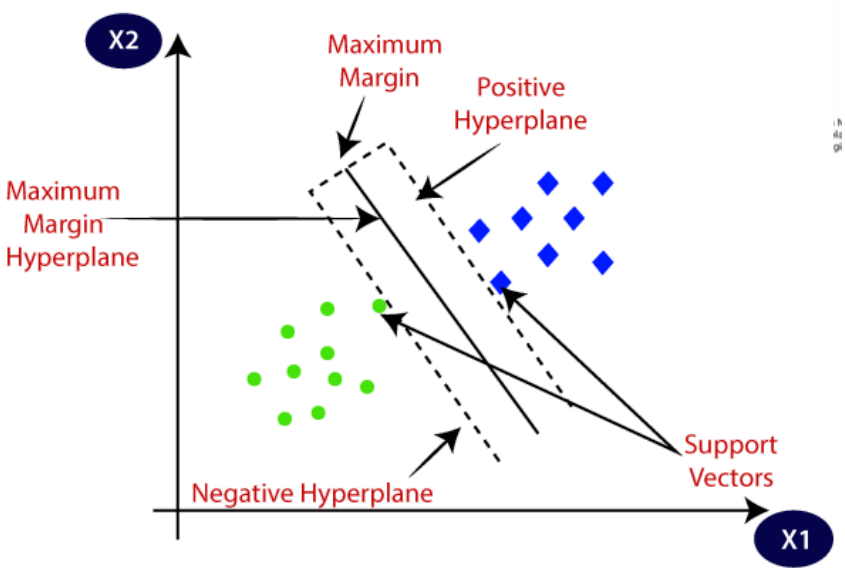
$$\text{Minimize}_{(w,b)} \quad \frac{\|w\|}{2} + C \sum_{i=1}^{n} |\eta_i|$$



$C = 4$ (η_i)

Support Vector Machine Algorithm

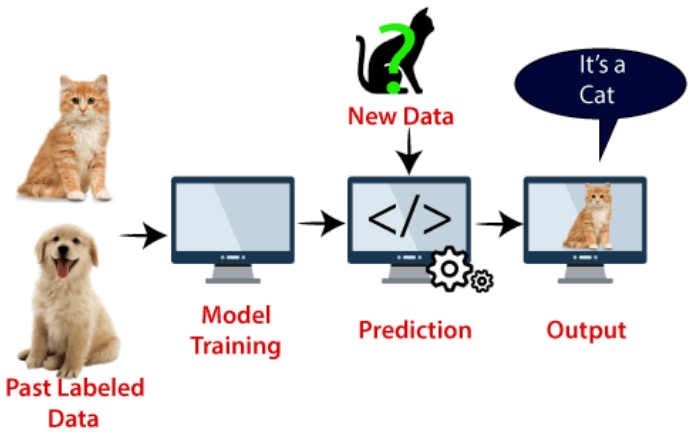
- Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems.
- However, primarily, it is used for Classification problems in Machine Learning.
- The goal of the SVM algorithm is to create the best line or decision 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.

- Consider the below diagram:



SVM algorithm can be used for Face detection, image classification, text categorization, etc.

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:

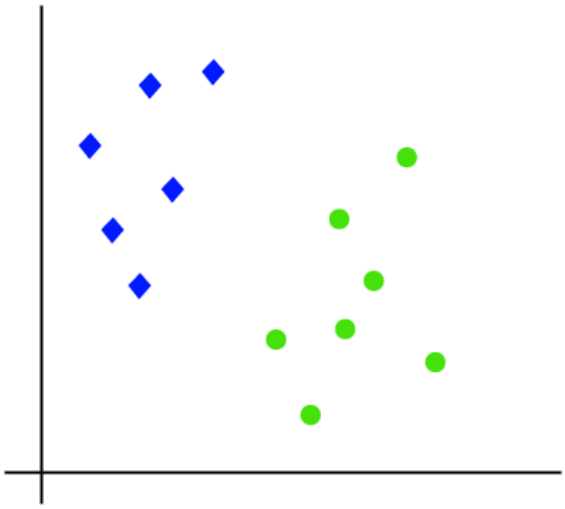
The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

How does SVM works?

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.

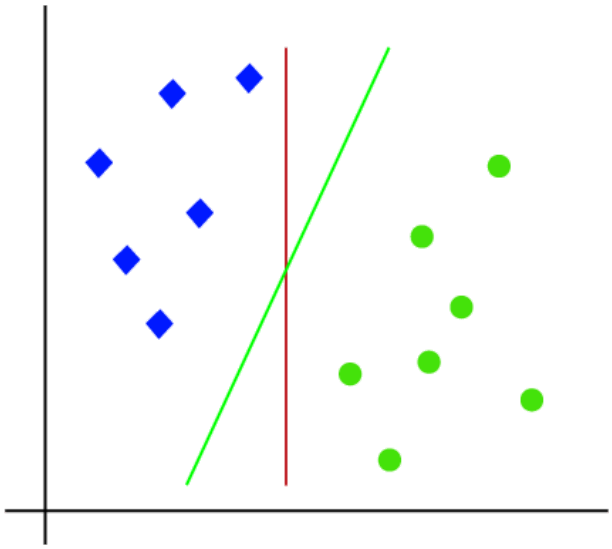
We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue.

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.**

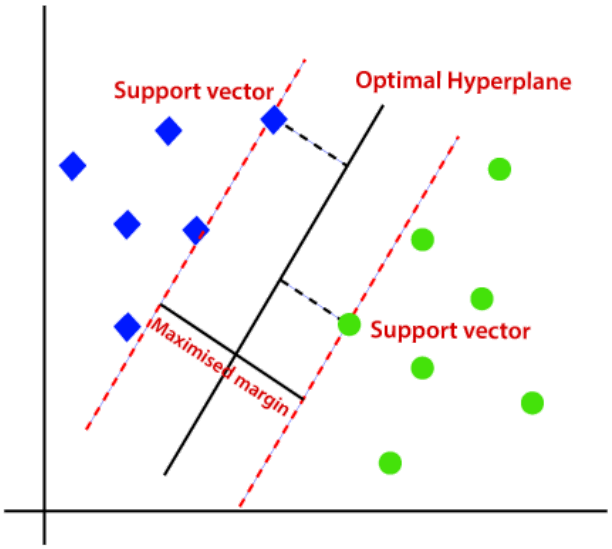
SVM algorithm finds the closest point of the lines from both the classes.

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.

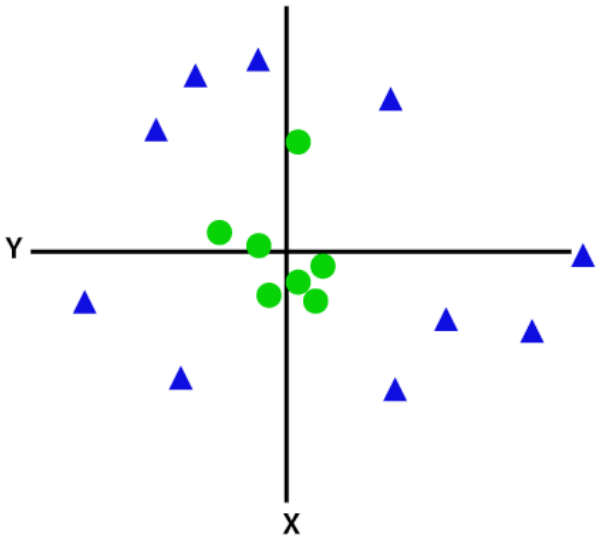
The **hyperplane** with maximum margin is called the **optimal hyperplane.**



Non-Linear SVM:

Non-Linear SVM:

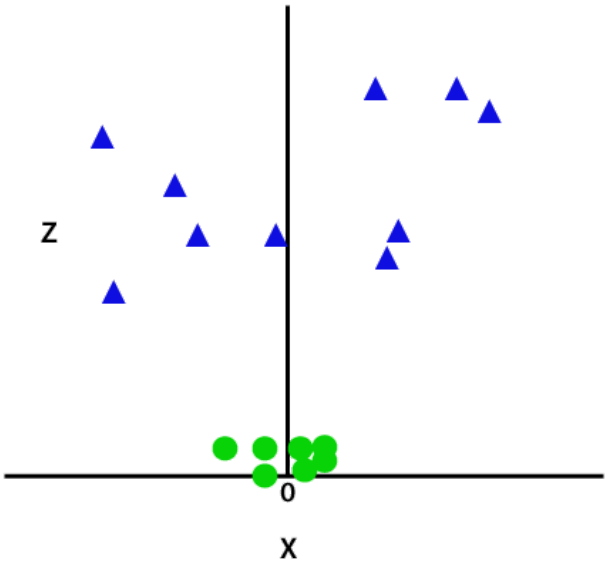
If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:



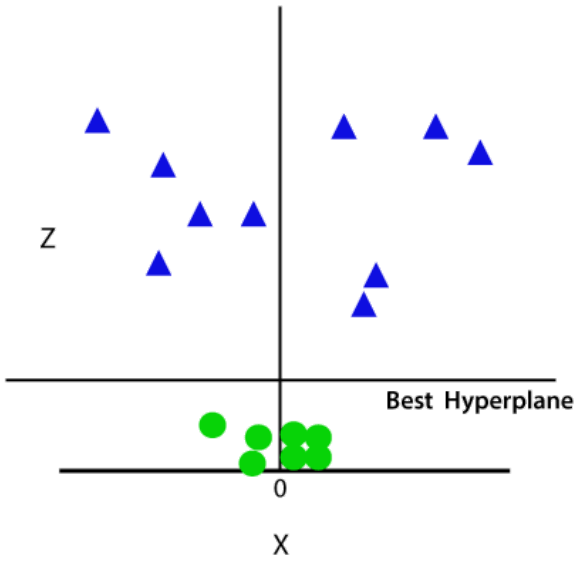
So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

z=x2 +y2

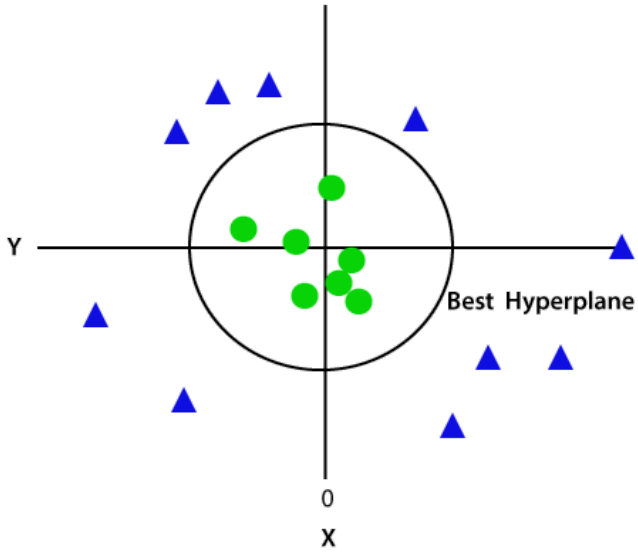
By adding the third dimension, the sample space will become as below image:



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:



Python Implementation of Support Vector Machine

Now we will implement the SVM algorithm using Python. Here we will use the same dataset user_data, which we have used in Logistic regression.

- **Data Pre-processing step**

In [1]:

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

In [2]:

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

Out[2]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

In [3]:

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

In [4]:

```
# 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)
```

In [5]:

```
#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)
```

After executing the above code, we will pre-process the data. The code will give the dataset as:

In [22]:

```
data_set
```

Out[22]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

Fitting the SVM classifier to the training set:

Now the training set will be fitted to the SVM classifier.

To create the SVM classifier, we will import SVC class from Sklearn.svm library. Below is the code for it:

In [6]:

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

Out[6]:

```
SVC(kernel='linear', random_state=0)
```

In the above code, we have used

kernel='linear',

as here we are creating SVM for linearly separable data.

However, we can change it for non-linear data.

And then we fitted the classifier to the training dataset(x_train, y_train)

Predicting the test set result:

Now, we will predict the output for test set. For this, we will create a new vector y_pred. Below is the code for it:

In [7]:

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

In [8]:

```
y_test
```

Out[8]:

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

In [9]:

```
y_pred
```

Out[9]:

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

Creating the confusion matrix:

Now we will see the performance of the SVM classifier that how many incorrect predictions are there as compared to the Logistic regression classifier.

To create the confusion matrix, we need to import the confusion_matrix function of the sklearn library.

After importing the function, we will call it using a new variable cm. The function takes two parameters, mainly y_true(the actual values) and y_pred (the targeted value return by the classifier).

Below is the code for it:

In [10]:

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

In [11]:

```
cm
```

Out[11]:

```
array([[66,  2],
       [ 8, 24]], dtype=int64)
```

As we can see in the above output image, there are 66+24= 90 correct predictions and 8+2= 10 incorrect predictions.

Support Vector Regression (SVR)

In [12]:

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

In [13]:

```
dataset=pd.read_csv('Position_Salaries.csv')
```

In [14]:

```
dataset.head()
```

Out[14]:

	Position	Level	Salary
0	Business Analyst	1	45000
1	Junior Consultant	2	50000
2	Senior Consultant	3	60000
3	Manager	4	80000
4	Country Manager	5	110000

In [15]:

```
x=dataset.iloc[:,1:2].values
y=dataset.iloc[:,2:].values
```

In [16]:

```
from sklearn.preprocessing import StandardScaler
```

In [17]:

```
st_x=StandardScaler()
st_y=StandardScaler()
```

In [18]:

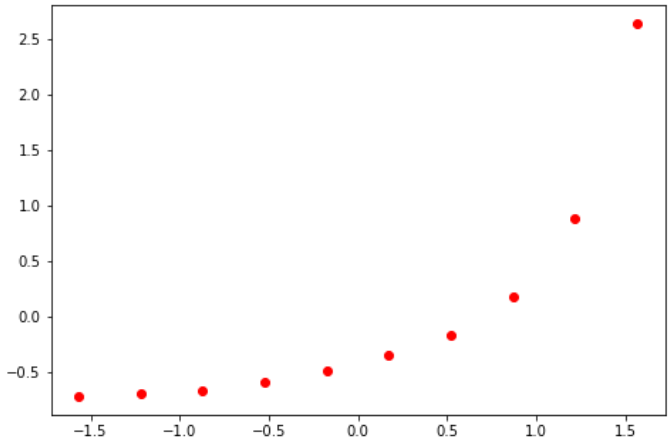
```
X=st_x.fit_transform(x)
Y=st_y.fit_transform(y)
```

In [19]:

```
fig=plt.figure()
ax=fig.add_axes([0,0,1,1])
ax.scatter(X,Y,color='r')
```

Out[19]:

```
<matplotlib.collections.PathCollection at 0x1f923d047c0>
```



In [21]:

```
from sklearn.svm import SVR
```

In [22]:

```
regressor=SVR(kernel='rbf')
```

In [23]:

```
regressor.fit(X,Y)
```

```
C:\Users\Shreeji\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vecto
r y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
```

Out[23]:

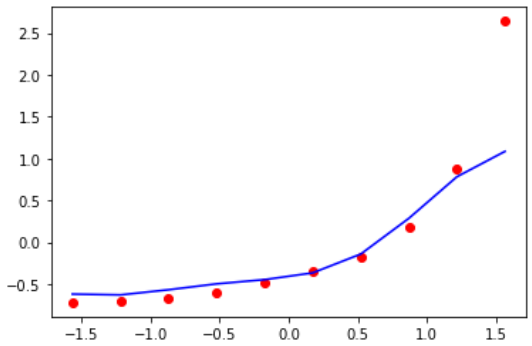
```
SVR()
```

In [24]:

```
plt.scatter(X,Y,color='red')
plt.plot(X,regressor.predict(X),color='blue')
```

Out[24]:

```
[<matplotlib.lines.Line2D at 0x1f924568b20>]
```



Choosing Parameters using Grid Search-SVM

In [10]:

```
import pandas as pd
import numpy as np
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix ,accuracy_score
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:

```
irisdata = pd.read_csv('Iris.csv')
```

In [3]:

```
irisdata.head()
```

Out[3]:

	sepal_length	sepal_width	petal_length	petal_width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

In [4]:

```
irisdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   class         150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

In [5]:

```
from sklearn.model_selection import train_test_split
X = irisdata.drop('class', axis=1)
y = irisdata['class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
```

In [6]:

```
#Apply kernels to transform the data to a higher dimension
kernels = ['Polynomial', 'RBF', 'Sigmoid','Linear']#A function which returns the corresponding SVC model
def getClassifier(ktype):
    if ktype == 0:
        # Polynomial kernal
        return SVC(kernel='poly', degree=8, gamma="auto")
    elif ktype == 1:
        # Radial Basis Function kernal
        return SVC(kernel='rbf', gamma="auto")
    elif ktype == 2:
        # Sigmoid kernal
        return SVC(kernel='sigmoid', gamma="auto")
    elif ktype == 3:
        # Linear kernal
        return SVC(kernel='linear', gamma="auto")
```

In [14]:

```
#Train a model
#Now it's time to train a Support Vector Machine Classifier.
#Call the SVC() model from sklearn and fit the model to the training data
```

In [11]:

```
for i in range(4):
    # Separate data into test and training sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)# Train a SVC model using different kernal
    svclassifier = getClassifier(i)
    svclassifier.fit(X_train, y_train)# Make prediction
    y_pred = svclassifier.predict(X_test)# Evaluate our model
    print("Evaluation:", kernels[i], "kernel")
    #print(classification_report(y_test,y_pred))
    print(accuracy_score(y_test,y_pred))
```

```
Evaluation: Polynomial kernel
0.9666666666666667
Evaluation: RBF kernel
1.0
Evaluation: Sigmoid kernel
0.26666666666666666
Evaluation: Linear kernel
0.9
```

Tuning the hyper-parameters of an estimator

Hyper-parameters are parameters that are not directly learnt within estimators.

In scikit-learn, they are passed as arguments to the constructor of the estimator classes.

Grid search is commonly used as an approach to hyper-parameter tuning that will methodically build and evaluate a model for each combination of algorithm parameters specified in a grid.

In [14]:

```
from sklearn.model_selection import GridSearchCV
```

In [15]:

```
param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly', 'sigmoid']}
```

In [16]:

```
grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=2)
grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
[CV] END .........................C=0.1, gamma=1, kernel=rbf; total time=   0.0s
[CV] END .........................C=0.1, gamma=1, kernel=rbf; total time=   0.0s
[CV] END .........................C=0.1, gamma=1, kernel=rbf; total time=   0.0s
[CV] END .........................C=0.1, gamma=1, kernel=rbf; total time=   0.0s
[CV] END .........................C=0.1, gamma=1, kernel=rbf; total time=   0.0s
[CV] END ........................C=0.1, gamma=1, kernel=poly; total time=   0.0s
[CV] END ........................C=0.1, gamma=1, kernel=poly; total time=   0.0s
[CV] END ........................C=0.1, gamma=1, kernel=poly; total time=   0.0s
[CV] END ........................C=0.1, gamma=1, kernel=poly; total time=   0.0s
[CV] END ........................C=0.1, gamma=1, kernel=poly; total time=   0.0s
[CV] END .......................C=0.1, gamma=1, kernel=sigmoid; total time=   0.0s
[CV] END .......................C=0.1, gamma=1, kernel=sigmoid; total time=   0.0s
[CV] END .......................C=0.1, gamma=1, kernel=sigmoid; total time=   0.0s
[CV] END .......................C=0.1, gamma=1, kernel=sigmoid; total time=   0.0s
[CV] END .......................C=0.1, gamma=1, kernel=sigmoid; total time=   0.0s
[CV] END .......................C=0.1, gamma=0.1, kernel=rbf; total time=   0.0s
[CV] END .......................C=0.1, gamma=0.1, kernel=rbf; total time=   0.0s
[CV] END .......................C=0.1, gamma=0.1, kernel=rbf; total time=   0.0s
```

In [17]:

```
#Find the optimal parameters

print(grid.best_estimator_)
```

```
SVC(C=100, gamma=0.01)
```

found the best estimator using grid search Take this grid model to create some predictions using the test set and then create classification reports and confusion matrices

In [18]:

```
grid_predictions = grid.predict(X_test)
print(confusion_matrix(y_test,grid_predictions))
print(classification_report(y_test,grid_predictions))
```

```
[[12  0  0]
 [ 0  9  3]
 [ 0  0  6]]
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        12
Iris-versicolor       1.00      0.75      0.86        12
 Iris-virginica       0.67      1.00      0.80         6

       accuracy                           0.90        30
      macro avg       0.89      0.92      0.89        30
   weighted avg       0.93      0.90      0.90        30
```

In [ ]:

K-Nearest Neighbor(KNN) Algorithm for Machine Learning

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- 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.
- 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.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.

Support Vector Machines

Support Vector Classification

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

```
data_set= pd.read_csv('User_Data.csv')
data_set.head()
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

```
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values
```

```
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)
```

```
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" Support vector
classifier = SVC(kernel='linear', random_state=0)
classifier.fit(x_train, y_train)
```

```
SVC(kernel='linear', random_state=0)
```

```
y_pred= classifier.predict(x_test)
```

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm= confusion_matrix(y_test, y_pred)
print(accuracy_score(y_test, y_pred))
cm
```

```
0.9
array([[66,  2],
       [ 8, 24]])
```

```
1
```

Support Vector Regression

```
dataset=pd.read_csv('Position_Salaries.csv')
```

```
dataset.head()
```

```
Position  Level  Salary
```

```
x=dataset.iloc[:,1:2].values
y=dataset.iloc[:,2:].values
```

```
1  Junior Consultant       2   50000
```

```
from sklearn.preprocessing import StandardScaler
```

```
3          Manager       4   80000
```

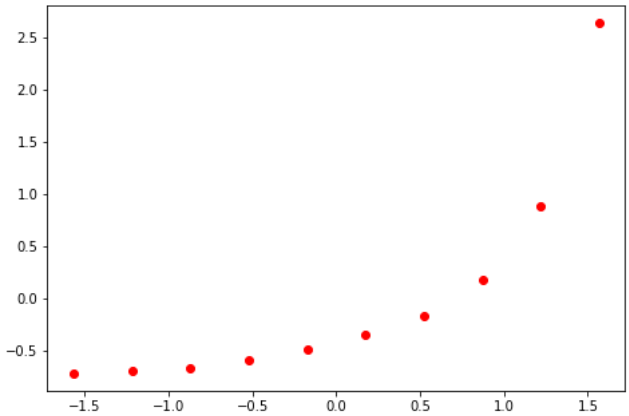
```
st_x=StandardScaler()
st_y=StandardScaler()
X=st_x.fit_transform(x)
Y=st_y.fit_transform(y)

```

```
from matplotlib import pyplot as plt
fig=plt.figure()
ax=fig.add_axes([0,0,1,1])
ax.scatter(X,Y,color='r')

```

```
<matplotlib.collections.PathCollection at 0x7f6ec164e910>
```

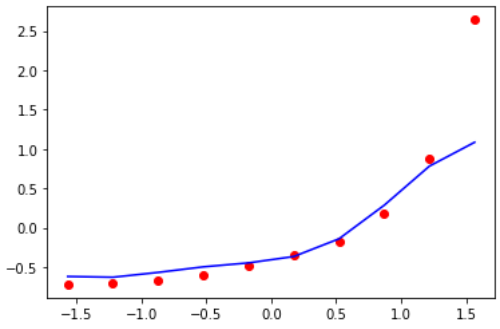



```
from sklearn.svm import SVR
regressor=SVR(kernel='rbf')
regressor.fit(X,Y)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/utils/validation.py:993: DataConversionWarning: A column-vector y was passed when a
  y = column_or_1d(y, warn=True)
SVR()
```

```
plt.scatter(X,Y,color='red')
plt.plot(X,regressor.predict(X),color='blue')
```

```
[<matplotlib.lines.Line2D at 0x7f6ec1152130>]
```




```
1
```

Choosing Parameters using Grid SearchSVM

```
import pandas as pd
import numpy as np
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix ,accuracy_score
import matplotlib.pyplot as plt
%matplotlib inline

```

```
irisdata = pd.read_csv('Iris.csv')
```

```
irisdata.head()
```

	sepal_length	sepal_width	petal_length	petal_width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
from sklearn.model_selection import train_test_split
X = irisdata.drop('class', axis=1)
y = irisdata['class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
```

```
kernels = ['Polynomial', 'RBF', 'Sigmoid','Linear']#A function which returns the co
def getClassifier(ktype):
 if ktype == 0:
 # Polynomial kernal
   return SVC(kernel='poly', degree=8, gamma="auto")
 elif ktype == 1:
 # Radial Basis Function kernal
  return SVC(kernel='rbf', gamma="auto")
 elif ktype == 2:
 # Sigmoid kernal
  return SVC(kernel='sigmoid', gamma="auto")
 elif ktype == 3:
 # Linear kernal
  return SVC(kernel='linear', gamma="auto")
```

```
for i in range(4):
 # Separate data into test and training sets
 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)# Tr
 svclassifier = getClassifier(i)
 svclassifier.fit(X_train, y_train)# Make prediction
 y_pred = svclassifier.predict(X_test)# Evaluate our model
 print("Evaluation:", kernels[i], "kernel")
 #print(classification_report(y_test,y_pred))
 print(accuracy_score(y_test,y_pred))
```

```
Evaluation: Polynomial kernel
0.9666666666666667
Evaluation: RBF kernel
1.0
Evaluation: Sigmoid kernel
0.26666666666666666
Evaluation: Linear kernel
0.9666666666666667
```

```
from sklearn.model_selection import GridSearchCV
```

```

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf','sigmoid','poly']}
```

```
grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=2)
grid.fit(X_train,y_train)
```

```
[CV] END .....................C=100, gamma=1, kernel=poly; total time=   0.0s
[CV] END .....................C=100, gamma=0.1, kernel=rbf; total time=   0.0s
[CV] END .....................C=100, gamma=0.1, kernel=rbf; total time=   0.0s
[CV] END .....................C=100, gamma=0.1, kernel=rbf; total time=   0.0s
[CV] END .....................C=100, gamma=0.1, kernel=rbf; total time=   0.0s
[CV] END .....................C=100, gamma=0.1, kernel=rbf; total time=   0.0s
[CV] END ................C=100, gamma=0.1, kernel=sigmoid; total time=   0.0s
[CV] END ................C=100, gamma=0.1, kernel=sigmoid; total time=   0.0s
[CV] END ................C=100, gamma=0.1, kernel=sigmoid; total time=   0.0s
[CV] END ................C=100, gamma=0.1, kernel=sigmoid; total time=   0.0s
[CV] END ................C=100, gamma=0.1, kernel=sigmoid; total time=   0.0s
[CV] END ....................C=100, gamma=0.1, kernel=poly; total time=   0.0s
[CV] END ....................C=100, gamma=0.1, kernel=poly; total time=   0.0s
[CV] END ....................C=100, gamma=0.1, kernel=poly; total time=   0.0s
[CV] END ....................C=100, gamma=0.1, kernel=poly; total time=   0.0s
[CV] END ....................C=100, gamma=0.1, kernel=poly; total time=   0.0s
[CV] END ....................C=100, gamma=0.01, kernel=rbf; total time=   0.0s
[CV] END ....................C=100, gamma=0.01, kernel=rbf; total time=   0.0s
[CV] END ....................C=100, gamma=0.01, kernel=rbf; total time=   0.0s
[CV] END ....................C=100, gamma=0.01, kernel=rbf; total time=   0.0s
[CV] END ....................C=100, gamma=0.01, kernel=rbf; total time=   0.0s
[CV] END ................C=100, gamma=0.01, kernel=sigmoid; total time=   0.0s
[CV] END ................C=100, gamma=0.01, kernel=sigmoid; total time=   0.0s
[CV] END ................C=100, gamma=0.01, kernel=sigmoid; total time=   0.0s
[CV] END ................C=100, gamma=0.01, kernel=sigmoid; total time=   0.0s
[CV] END ................C=100, gamma=0.01, kernel=sigmoid; total time=   0.0s
[CV] END ....................C=100, gamma=0.01, kernel=poly; total time=   0.0s
[CV] END ....................C=100, gamma=0.01, kernel=poly; total time=   0.0s
[CV] END ....................C=100, gamma=0.01, kernel=poly; total time=   0.0s
[CV] END ....................C=100, gamma=0.01, kernel=poly; total time=   0.0s
[CV] END ....................C=100, gamma=0.01, kernel=poly; total time=   0.0s
[CV] END ...................C=100, gamma=0.001, kernel=rbf; total time=   0.0s
[CV] END ...................C=100, gamma=0.001, kernel=rbf; total time=   0.0s
[CV] END ...................C=100, gamma=0.001, kernel=rbf; total time=   0.0s
[CV] END ...................C=100, gamma=0.001, kernel=rbf; total time=   0.0s
[CV] END ...................C=100, gamma=0.001, kernel=rbf; total time=   0.0s
[CV] END ...............C=100, gamma=0.001, kernel=sigmoid; total time=   0.0s
[CV] END ...............C=100, gamma=0.001, kernel=sigmoid; total time=   0.0s
[CV] END ...............C=100, gamma=0.001, kernel=sigmoid; total time=   0.0s
[CV] END ...............C=100, gamma=0.001, kernel=sigmoid; total time=   0.0s
[CV] END ...............C=100, gamma=0.001, kernel=sigmoid; total time=   0.0s
[CV] END ..................C=100, gamma=0.001, kernel=poly; total time=   0.0s
[CV] END ..................C=100, gamma=0.001, kernel=poly; total time=   0.0s
[CV] END ..................C=100, gamma=0.001, kernel=poly; total time=   0.0s
[CV] END ..................C=100, gamma=0.001, kernel=poly; total time=   0.0s
[CV] END ..................C=100, gamma=0.001, kernel=poly; total time=   0.0s
GridSearchCV(estimator=SVC(),
             param_grid={'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01, 0.001],
                         'kernel': ['rbf', 'sigmoid', 'poly']},
             verbose=2)
```

```
1 grid.best_params_
```

```
{'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}
```

```
1 grid_predictions = grid.predict(X_test)
2 print(confusion_matrix(y_test,grid_predictions))
3 print(classification_report(y_test,grid_predictions))
```

```
[[ 7  0  0]
 [ 0 14  1]
 [ 0  0  8]]
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       1.00      0.93      0.97        15
 Iris-virginica       0.89      1.00      0.94         8

       accuracy                           0.97        30
      macro avg       0.96      0.98      0.97        30
   weighted avg       0.97      0.97      0.97        30
```

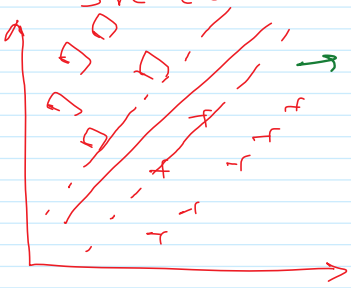
```
1
```

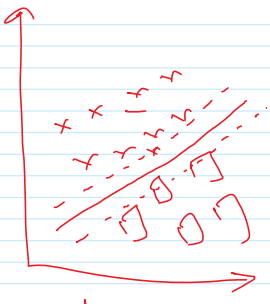
Non Linear SVM

19 December 2022 10:09 AM

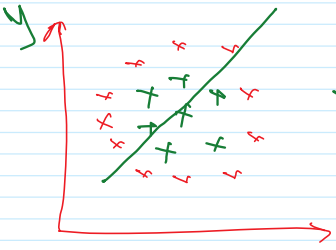
SVM < SVC / SVR

SVC (Linear SVC) → Linearly Seperable → Kernel = "linear" → SVC (Linear SVC)





↓ linear SVM



→ Linearly Seperable ✗

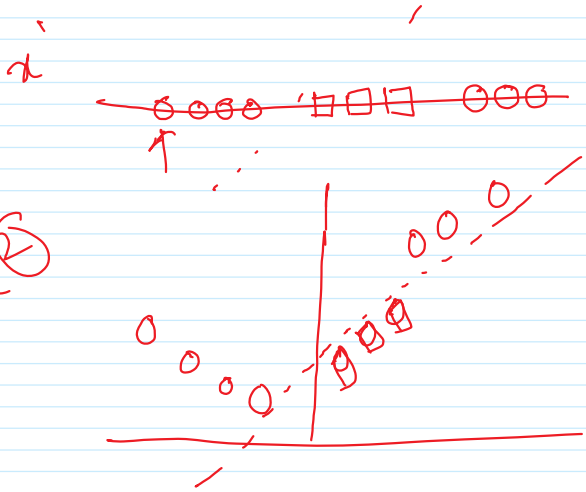
↳ If Data is Not linearly seperable than use Non linear SVM & kernel fn:

↳ Transformation by using Mathematical formulae

Low Dimension Data → Kernel → High Dimension Data

x | x^2

E.g:



1-D Data

class - ○

class - □

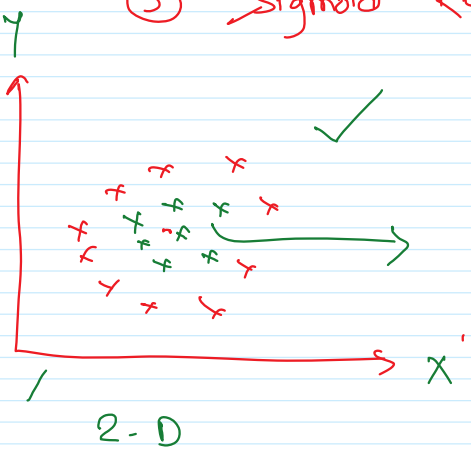
↳ 2-D ply kernel =

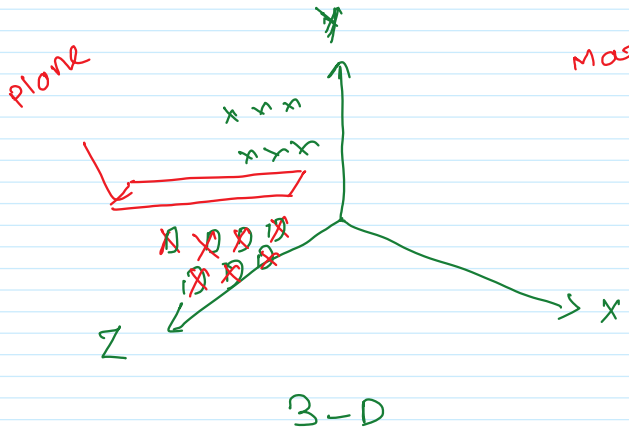
Types of kernel: → kernel = f(x)

① Rbf kernel (Radial basis fn)

② Polynomial kernel

③ Sigmoid kernel





Mathematical formula

↳ Grid search cv