2025

Basic Data Science

A Complete Guide

Kushal Prasad Joshi

kushalprasadjoshi@gmail.com

Preface

Welcome to this handwritten guide, a resource created to assist learners, practitioners, and enthusiasts in the field of Data Science. This guide is the result of my personal exploration and understanding of Data Science and Machine Learning, developed outside of any formal course structure. It aims to provide accessible, practical knowledge for anyone eager to explore and deepen their understanding of this subject.

Throughout the creation of this note, I have drawn inspiration from a variety of sources, including textbooks, online tutorials and real-world applications. This material is suitable for learners at beginner level. Each chapter is structured to provide clear explanations, relevant examples and exercises that encourage active learning and problem solving.

Thank you for choosing this resource. I hope it proves valuable as you explore, learn and grow.

Happy Learning!

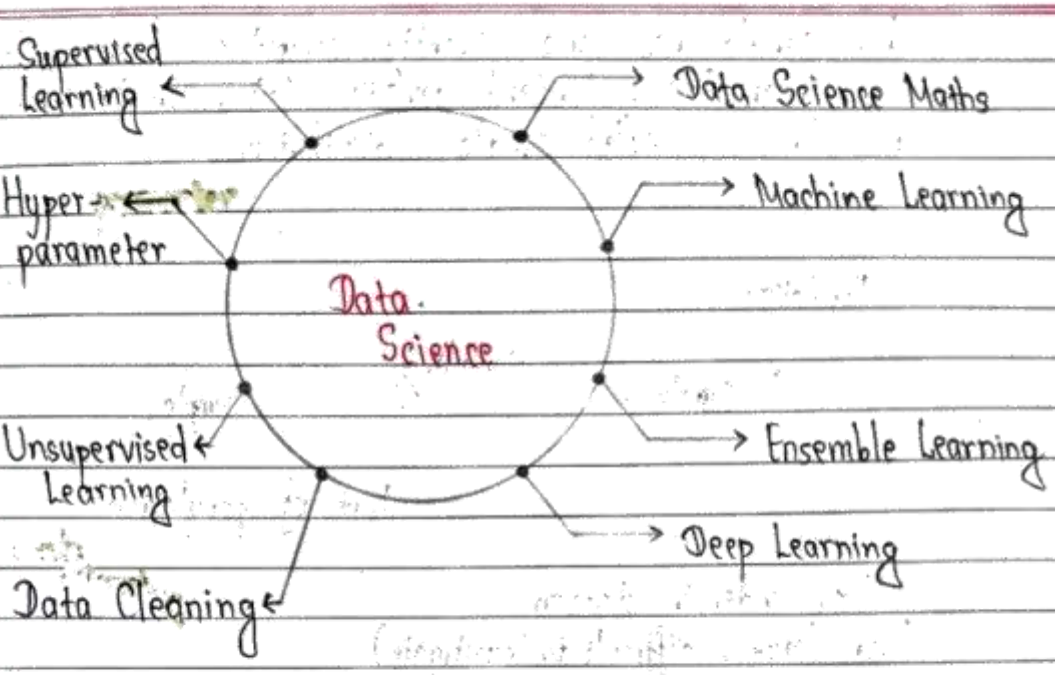
Kushal Prasad Joshi
kushalprasadjoshi@gmail.com
Author

Table of Contents

If you get on the wrong train
Make sure to get off the nearest station
The longer it takes you to get off
The more expensive the return trip will be.

- Japanese Quote

1. Introduction



1. Population and Sample Data

Population (N):

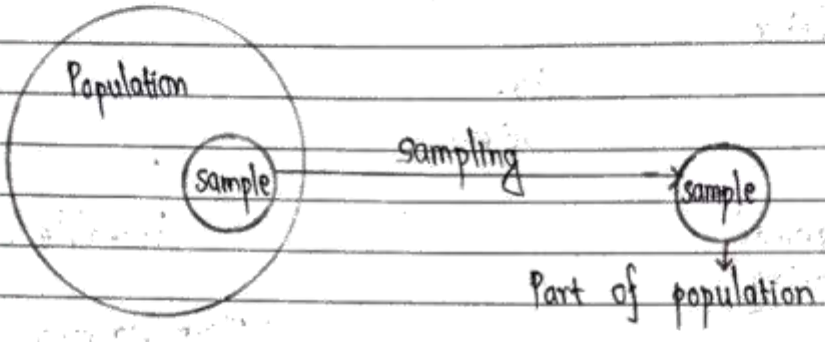
A population includes all of the elements from a set of data. The population is a whole set of values or individuals, you are interested in.

Population characteristics are mean (μ), standerd deviation (σ), propartion (P), median, percentiles, etc. The value of a population characteristics is fixed. These characteristics are called population distribution.

Sample (n):

The sample is subset of population and is the set of values you actually use in your estimation.

This sample has some quantity computed from values e.g. mean ($\bar{x}$), standerd deviation (s), sample proportion, etc. This is called sample distribution.



Data under investigation
(very large, difficult to investigate)

2. Statistics

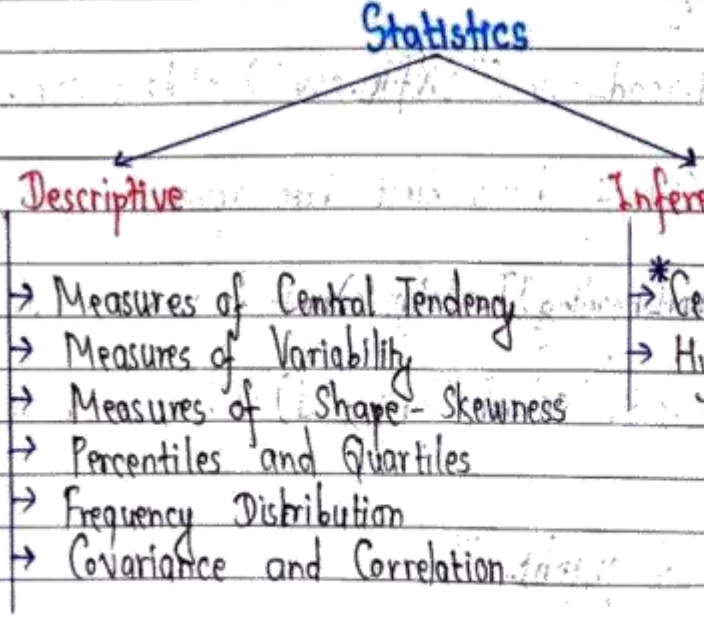
Date
Page 3

What is statistics?
Statistics is a set of mathematical methods and tools that enable us to answer important questions about data.

It is divided into two categories: Descriptive and Inferential

Descriptive Statistics: (on population data)
This offers methods to sumarized data by transforming raw observations into meaningful information that is easy to interpret and share.

Inferential Statistics: (on sample data → mostly used)
This offers methods to study experiments done on small samples of data chalk out the inferences to the enitre population (entire domain).



1. Measures of Central Tendency → mean, median, mode

1. Mean : (Average)

Mean = $\frac{\text{Sum of all data}}{\text{Total no. of data}}$

```
import numpy as np
import pandas as pd

ar = np.array([4,5,6,2,1,8,5,6,4,7])

np.sum(ar)/len(ar) # manual process

np.mean(ar) # automatic process

''' For solving real world problems. '''

dataset = pd.read_csv("Students.csv") # read csv file

dataset.head(3) # shows first three rows

dataset["Age"].mean() # find mean using pandas

np.mean(dataset["Age"]) # find mean using numpy

''' For graphical output '''
```

```
import matplotlib.pyplot as plt # to create a graphs
import seaborn as sns # for advancements in graphs

mn = np.mean(dataset["Age"]) # store mean in mn

sns.histplot(x="Age", data = dataset, bins = [i for i in
        range(0, 100, 10)])
 # bins is used to manage bins of graph columns.
 # This is column vs data graph.
plt.plot([mn for i in range(0, 250)], [i for i in
        range(0, 250)], c="red")
 # Display mean in graph.
plt.show() # display graph with all figures
```

2. Median : (Middle Value)

- Arrange data in ascending order.
- Odd no. of data → middle value
  Even no. of data → average of both middle values.

```
"" For finding median ""

dataset.isnull().sum() # displays total number of nulls in
                       # each columns in dataset.

dataset["Age"].fillna(dataset["Age"].mean(),
        inplace = True)
```

filling up the null values by mean of dataset column.
(data cleaning uses measurement of central tendency)

Now, there is no null value in column. So, can find median
np.median(dataset["Age"]) # find median using numpy

dataset["Age"].median() # find median using pandas

"" for Graphical output. ""

md = np.median(dataset["Age"]) # store median in md

sns.histplot(x="Age", data=dataset, bins=[i for i in range(0, 100, 10)]) # Graph format
plt.plot([md for i in range(0, 250)], [i for i in range(0, 250)], c="blue")
# Plot mean in graph.
plt.show() # Display graph with all figures

3. Mode : (Most Repeated Value)

- Mode is a data with highest frequency (repeatation).
- Commonly used in catagorical data. e.g. gender, class, material status, sex, etc.

"" for finding mode. ""

```
dataset = pd.read_csv("Students.csv")
dataset["Age"].mode()[0]  # Calculate mode using pandas.
    # index 0 is used because it returns a series and the
    # mode is the first data of series.

dataset["Age"].value_counts()  # Shows frequencies of
    # datas in column.


"""To get Graphical output."""

mo = dataset["Age"].mode()[0]  # Store mode in mo

sns.histplot(x="Age", data=dataset, bins=[i for i in
        range(0, 100, 10)])  # Graph format
plt.plot([mo for i in range(0, 250)], [i for i in range
        (0, 250)], c="green")  # Plot mode in graph
plt.show()  # Display graph with all figures


"""To display mean, median and mode in same graph"""

sns.histplot(x="column", data=dataset, bins=[i for i in
        range(0, 100, 10)])  # Graph format
plt.plot([mn for i in range(0, 250)], [i for i in range
        (0, 250)], c="red", label="mean")
    # Plot for mean in range(start, end)
    # label is used to show the labeling of colours with
    # the graphs in legend.
plt.plot([md for i in range(0, 250)], [i for i in range
        (0, 250)], c="blue", label="median")
```

```
# Plot median in graph
plt.plot([mo for i in range(0, 250)], [i for i in
    range(0, 250)], c="green", label="mode")
    # Plot mode in graph
plt.legend()  # Shows legend according to given labels
plt.show()  # Show graph with all figures
```

2. Measures of Variability

1. Range

- Range is the difference between the maximum and minimum values in the dataset.
  i.e. Range = maximum - minimum.
- It provides a simple measure of the spread of the data, but it can be sensitive to outliners.

```
''' for finding range of given dataset '''

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt  # not used
import seaborn as sns  # not used

dataset = pd.read_csv("Students.csv")

dataset.head(3)  # display first three rows.
```

```
min_r = dataset["Age"].min() # Returns min value
max_r = dataset["Age"].max() # Returns max value.

min_r, max_r  # Prints min and max values
              # Example: (0.42, 80.0)

range = max_r - min_r # Gives range of column data.
range  # Displays range
```

2. Mean Absolute Deviation

- The mean absolute deviation of a dataset is the average distance between each datapoint and the mean.
- It gives us an idea about the variability in dataset.
- Mathematically, $MAD = \frac{\sum |x_i - \bar{x}|}{n}$

"Making choice based on data of sections"

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

sec_a = np.array([75, 65, 73, 68, 72, 67])
sec_b = np.array([90, 47, 43, 96, 93, 51])
```

```
#Created two arrays with some data using numpy
no = np.array([1,2,3,4,5,6])
  # Array for y-axis values.

plt.figure(figsize=(10,3)) #Fix figure size
plt.scatter(sec_a, no, label="sec A")
    #Draw scatter graph of sec-a vs. no
plt.scatter(sec_b, no, c="blue", label="sec B")
    # Draw scatter graph of sec-b
plt.plot([70,70,70,70,70,70], no, c="red",
        label="mean")
    # plot a line of mean from 0 to 6.
plt.legend()  # display legend according to labels
plt.show()  # show graph with all figures.

''' From the digram, the less scattered data is less
deviated and should be choosen.
But sometimes drawing graphs can be a tedious
task or even not possible. So, we need to have some
mathematical calculations to meet our needs.

The calculations can be done as follows: '''

#Find mean of both sections
#Mean absolute division is used when we have got the
#same mean while figuring out for best.
#  i.e sec_a and sec-b has same mean
mean = np.mean(sec-b) #Same for sec-a.
mean #display mean  : 70.0

#Find mean absolute deviation for both.
```

```
mad_a = np.sum(np.abs(sec_a - mean)) / len(sec_a)
mad_b = np.sum(np.abs(sec_b - mean)) / len(sec_b)

mad_a, mad_b  # display mean absolute deviation:
              # (3.3333333333333335, 23.0)
```

"We found that mean absolute deviation of sec-a is less than mean standerd deviation of sec-b.
So, sec-a should be our choice."

"If we got same mean absolute deviation for both data then standard deviation is used."

3. Standerd Deviation

- The standerd deviation is the measure of the amount of variation or dispersion of a set of values.
- A low standerd deviation indicates the values tends to be close to mean (also called the expected value) of the set, while a high standerd deviation indicates that the values are spread out over a wider range.
- Mathematically, $\sigma = \sqrt{\frac{\sum (x_i - \mu)^2}{N}}$

where, σ = population standerd deviation.
N = the size of the population
x_i = each value from the population

μ = the population mean

"Calculating standerd deviation on previous lists."

np.std (sec-a), np.std (sec-b) # calculate and show:
(3.862.... , 23.180...)

4. Variance

- Variance is a measure of how data points differ from a mean.
- According to layman, a variance is a measure of how far a set of data (numbers) are spread out from their mean (average) value.
- Mathematically,

$$\text{Var} = \sigma^2 = \frac{\sum (x_i - \bar{x})^2}{N}$$

"Calculating variance on previous lists."

np.var (sec-a), np.var (sec-b) # calculate variance and
display: (14.91... , 537.33....)

"Calculating variance and standerd deviation on real

world data. If the variance and standerd deviation is high it is a scatterd data but as a data scientist, we always perfer less scattered data for our work."

Example :

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns  # import necessary libraries.

dataset = pd.read_csv("Students.csv")

dataset.head(3)

dataset["Age"].var()  # Calculate varrance.

dataset["Age"].std()  # Calculate standerd deviation.

# Pot a column vs data graph for visualization
sns.histplot(x="Age", data=dataset, bins=np.arrange(0,100,10))
plt.show()
```

" If we don't want them to calculate all we have another method which will show {count, mean, std, min, 25%, 50%, 75%, max} of all the numerical columns "

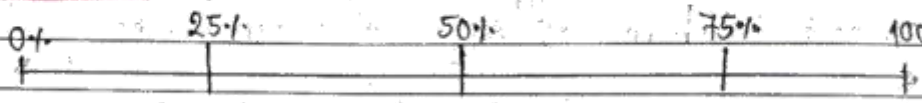
```
dataset.describe()
```

3. Percentiles and Quartiles

Percentiles

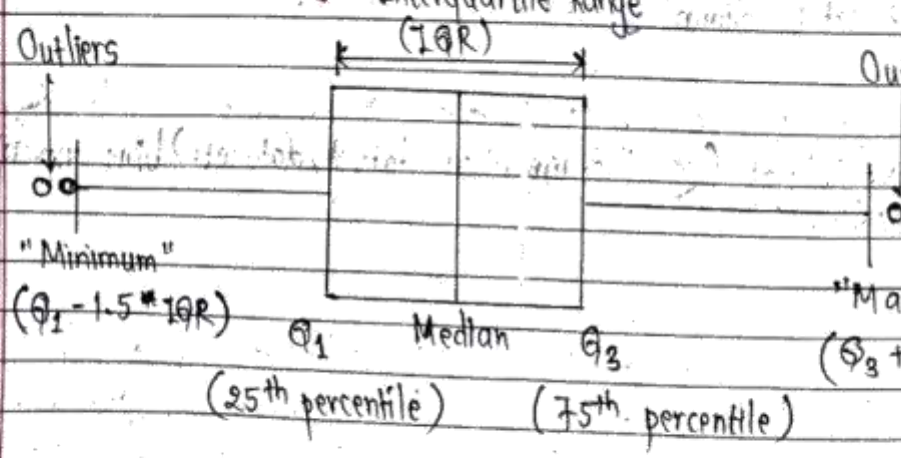
- Percentiles are used in statistics to give you a number that describes the value that a given percent of the values are lower than. It ranges from 0% to 100%.

Quartiles



- The equal four divisions of percentiles made are called quartiles.

Interquartile Range (IQR)



From diagram;

$$IQR = Q_3 - Q_1$$

NOTE:

- Percentage vs Percentile : Percentage works on fixed defined value whereas percentile works on undefined base value. Also, percentage shows obtained value whereas percentile shows obtained ranking.

- Outliers : The data beyond our range.

eg 2, 3, 4, 5, 6, 2, 3, 4, 5, 6, 100, 200, 500

data in range — outliers

From diagram; the outliers are values beyond Minimum and Maximum which can be calculated using IQR.

""" Using Percentiles and Quartiles to solve real world problems. """

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

dataset = pd.read_csv("Students.csv")

dataset.head(3) # Display first three rows

dataset.isnull().sum() # Display total number of null
    # values present in each column in dataset

dataset["Age"].fillna(dataset["Age"].mean(), inplace
    = True)
```

filling up the null values by mean of dataset
column.

Calculating percentiles.
np.percentile (dataset ["Age"], 50)
np.percentile (dataset ["Age"], 25)
np.percentile (dataset ["Age"], 75)

Displaying all percentiles 0%, 25%, 50%, 75%
and 100% of all numerical columns at once.
dataset.describe ()

"If you find a large gap between min and 25% or max and 75% there exist outliers. No. of outliers is directly proportional to the gap."

We can see outliers by using a box plot.
sns.boxplot (x="Age", data = dataset)
plt.show

"Percentiles are used to identify and remove outliers which working with machine learning algorithms."

4. Measures of Shape - Skewness

Skewness

↳ Skewness measures the asymmetry of the distribution.

- A skewness value of 0 indicates a perfectly symmetrical distribution.

- Positive skewness indicates that the distribution is skewed to the right (i.e. the tail is longer on the right), while negative skewness indicates a left skew (i.e. the tail is longer on the left).

- Mathematically,

$$\text{Skewness} = \frac{\sum (x_i - \bar{x})^3}{(N-1) \cdot \sigma^3}$$

Frequency and Cumulative Distribution

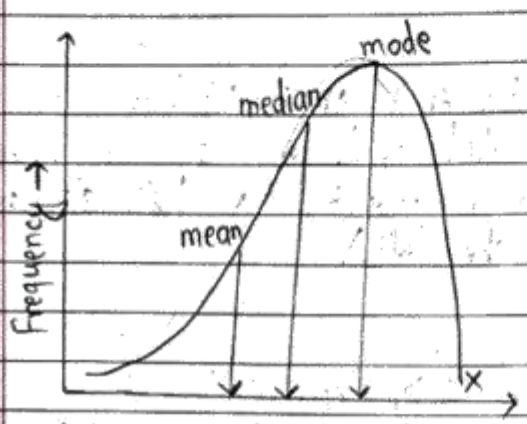
- A frequency distribution is a table that shows the number of occurances of different values in a dataset.

- A cumulative distribution shows the accumulation of frequencies up to a certain point. It is obtained by adding up the frequencies as you move through the values.

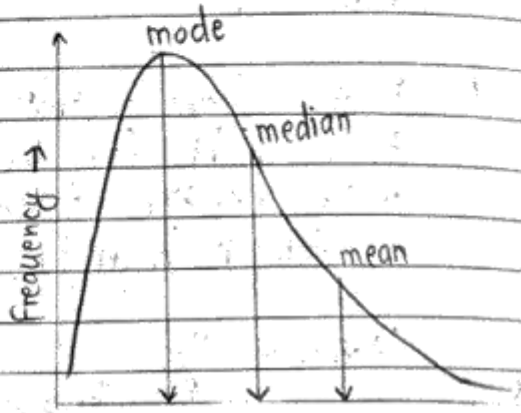
Value Range	Frequency	Cumulative
10 - 20	5	5
20 - 30	8	5+8 = 13
30 - 40	12	13+12 = 25
40 - 50	6	25+6 = 31

- Histogram is used for visualization.

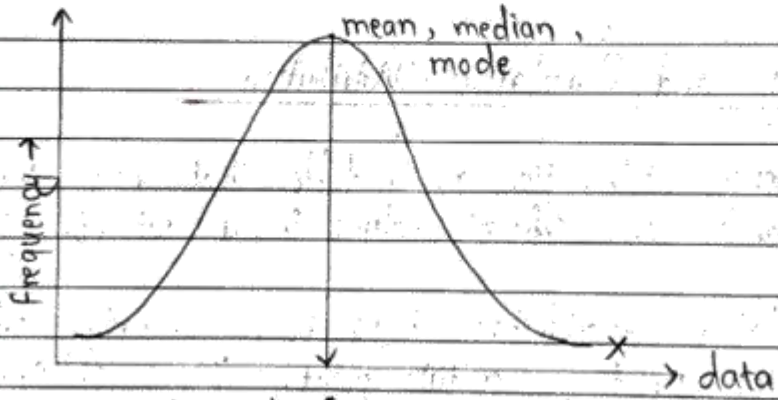
Skewness Charts.



(a) Negatively skewed



(b) Positively skewed.



(c) Normal (no skew)

From diagrams :

(a) When negatively skewed → mean < median < mode
(b) When positively skewed → mode < median < mean
(c) When no skew → mean = mode = median

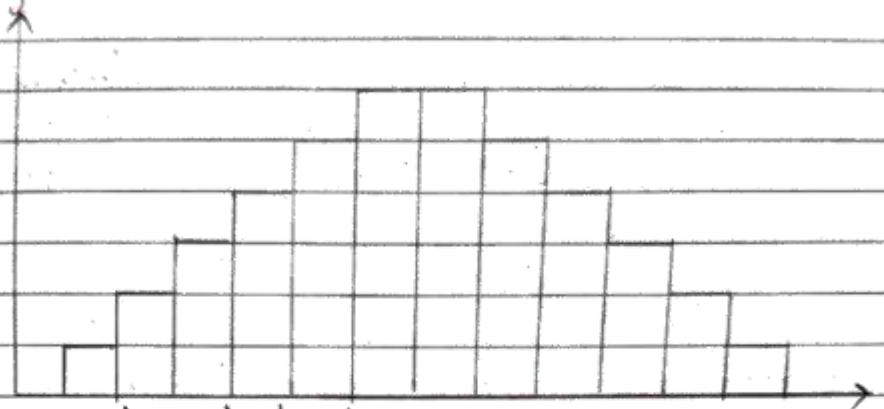
Working Practically in Jupyter :

```
import pandas as pd
```

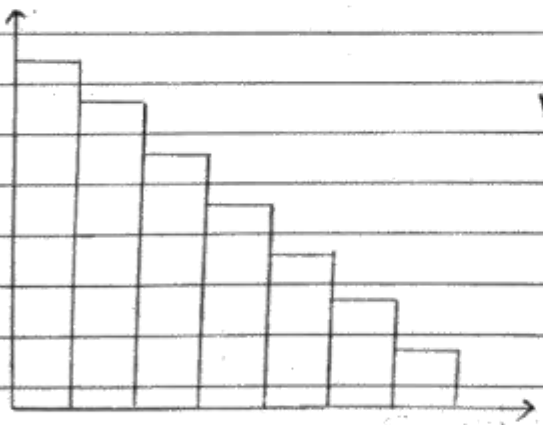
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

dataset = pd.read_csv("Students.csv")

dataset.head(3)

dataset["Age"].skew() # Returns skewness of column
''' If skewness > 0 → positively skewed
    If skewness < 0 → negatively skewed
    If skewness = 0 → no skew. '''

# Plot a histogram for visualization.
sns.histplot(x="Agen", data = dataset)
plt.show()


''' Visualization by creating your own data. '''

data = np.random.normal(0, 100, 100) # Create a random
                                     # array of data.

data # Display created data.

df = pd.DataFrame({"x": data}) # Create a column x where
                               # all data is placed in different rows.
df["x"].skew() # Gives skewness

# Create a histogram for visualization.
sns.histplot(x="x", data = df)
```

```
plt.show()

''' The normal non-skew data can be created as follows:'''

data = [2,3,3,4,4,4,5,5,5,5,6,6,6,6,6,7,7,7,7,7,
        7,7,8,8,8,8,8,9,9,9,9,10,10,10,11,
        11,12]  # non-skew data.

df = pd.DataFrame({"x": data}) # Create a column
   # named x under which all data is placed.

df["x"].skew()  # 0 skew

# Histogram will be symmetrical.
sns.histplot(x="x", data=df, bins=[2,3,4,5,6,7,8,9,
                                   10,11,12,13])
plt.show()

# Mean, Median, Mode will be same.
df["x"].mean(); df["x"].median(), df["x"].mode
        ()[0]    # (7.0, 7.0, 7)


''' Mean, Median and Modes will not be same in
negatively and positively skewed charts (data).
for negatively skewed: mean < median < mode
for positively skewed: mode < median < mean '''
```

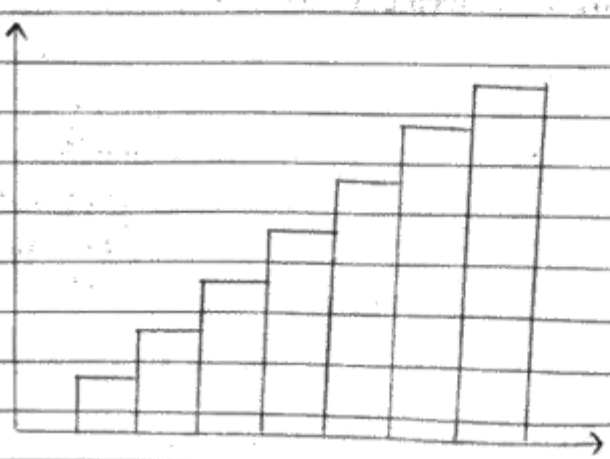
Symmetric (normal) vs skewed and normal distributions:



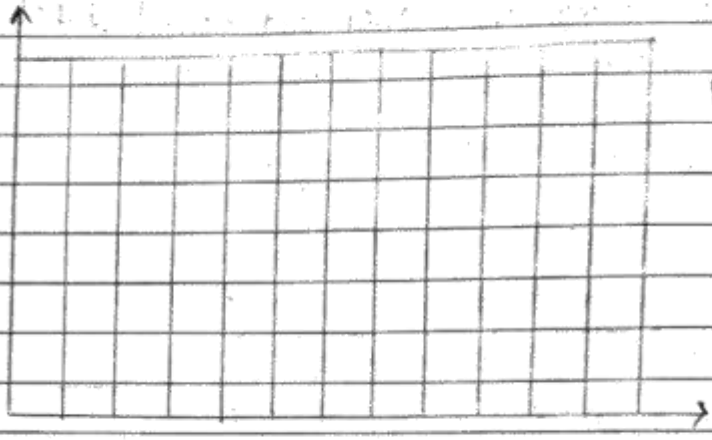
Normal distribution
(unimodal, symmetric, the "bell curve")



Right-skewed distribution
(Positively skewed)

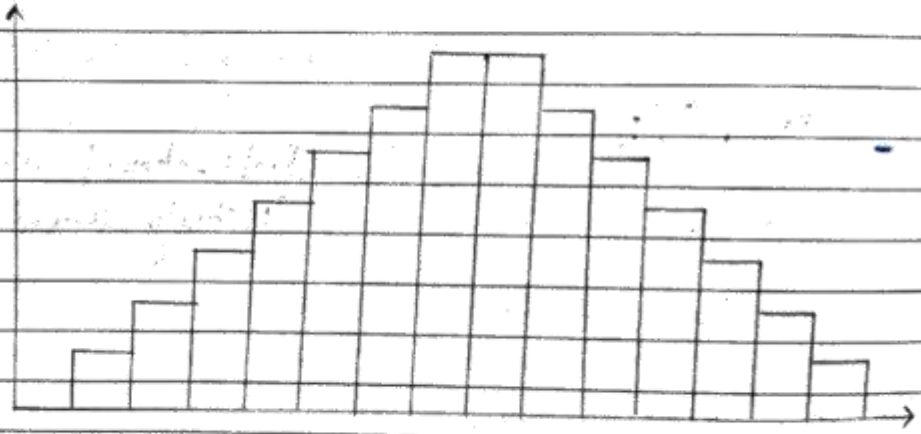


Left-skewed distribution
(Negatively skewed)

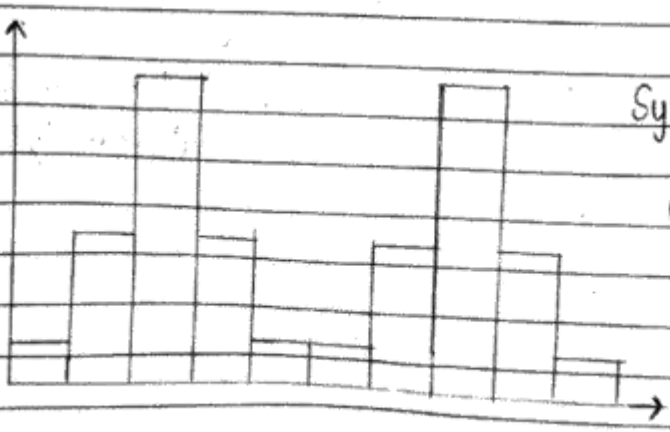


Uniform distribution
(equal spread,
no peaks)

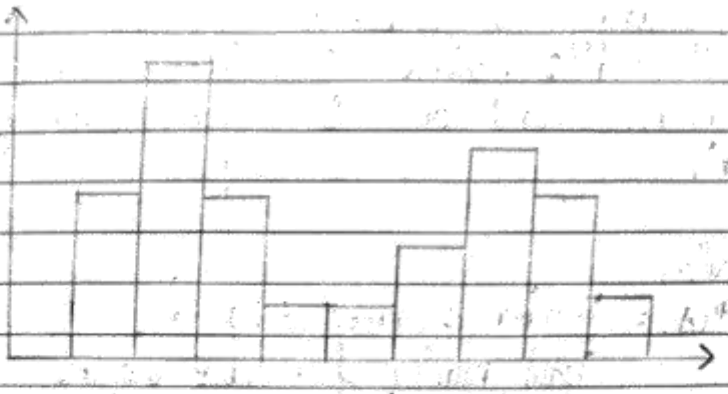
Unimodal vs Bimodel distributions.



Normal distribution
(unimodal, symmetric, the "bell curve")



Symmetrical bimodel
distribution
(two modes)



Non-symmetric bimodal distribution
(two nodes)

5. Probability

Random Variables : A Random variable x is a function that assigns a real number to each outcome in the sample space of a random experiment.

1. **Descrete Random Variable :** A random variable that takes on a countable number of distinct values. eg. dice roll

2. **Continuous Random Variable :** A random variable that can take an any value within a given range or interval. eg. height of people in a sample.
This doesn't have any fix no. of possible outcomes.

Probability

G Probability measures the likelihood of a particular outcome

or event occuring. It is typically expressed as a number between 0 and 1, where 0 indicates impossibility (event will not occur) and 1 indicates certainity (event will occur).

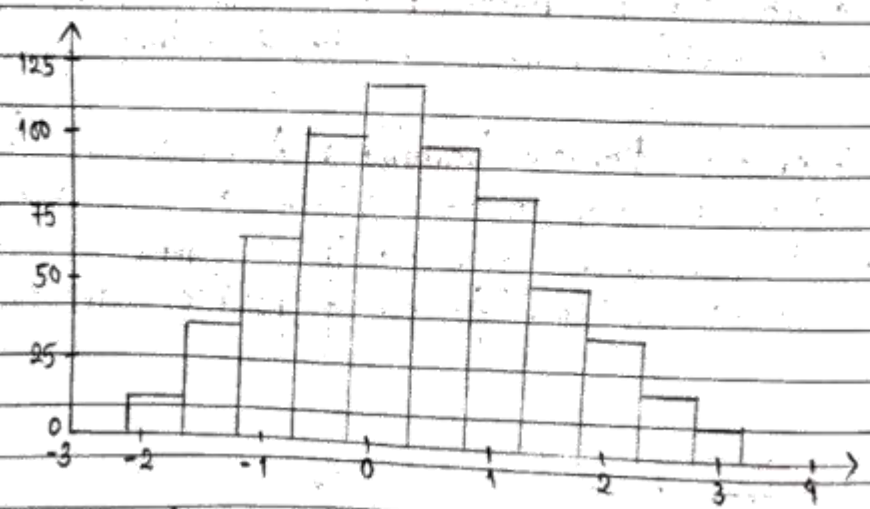
- Mathematically,

$$P(A) = \frac{\text{Number of times A occurs}}{\text{Total number of possible outcomes}}$$

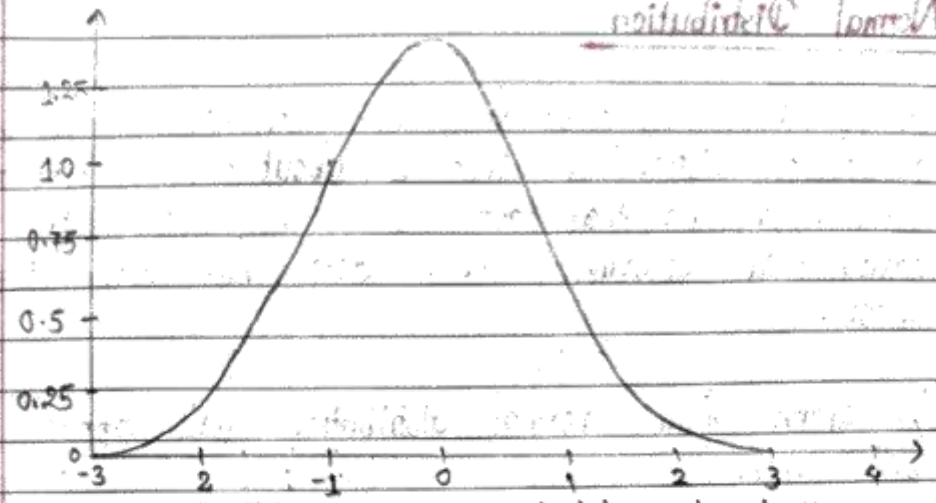
Probability Distribution

- Probability distributions describe how the probabilities of different outcomes are distributed over the sample space of a random variable.

- Two types of probability distribution are:
  (a) continuous probability distributions
  (b) descrete probability distributions



(a) Discrete probability distribution



(b) Continous probability distribution.

Probability Distribution Functions

- A probability distribution function is the mathematical function that gives the probabilities of occurrence of different possible outcomes for an experiment.

- Types of probability distribution functions are:
  (a) Probability density function (PDF)
  (b) Probability mass function (PMF)
  (c) Cumulative density function (CDF)

(a) **Probability density function (PDF)** : This function is defined on continous data. Density means total number of data in a particular range.

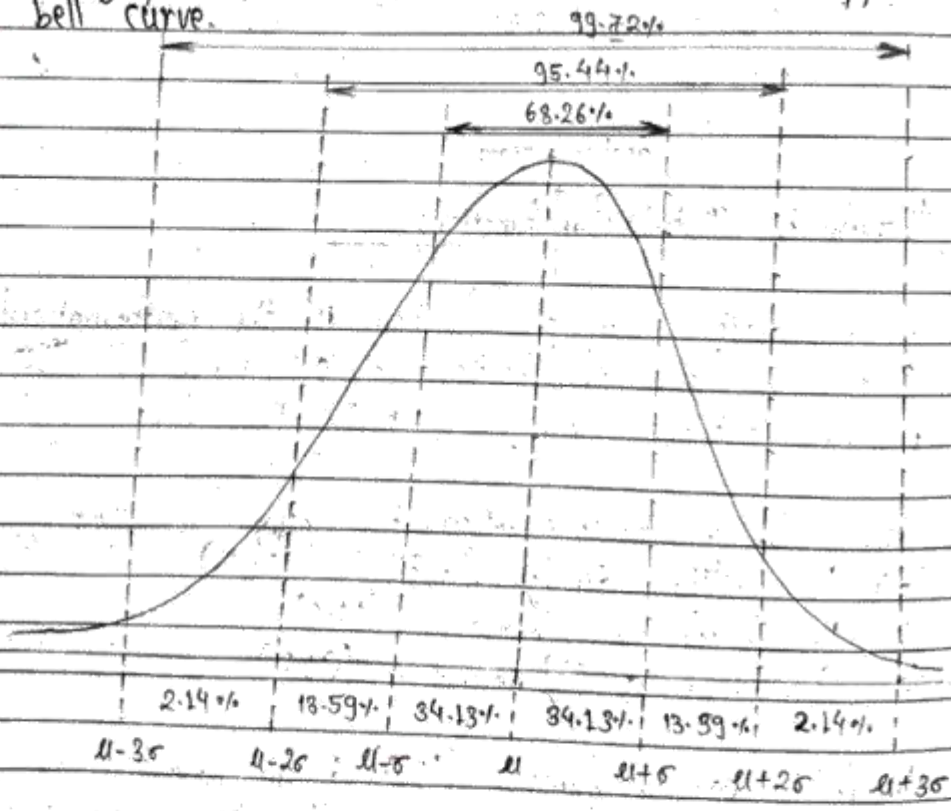
(b) **Probability mass function (PMF)** : This function is defined on descrete data. Mass means frequency of data.

(c) **Cumulative density function (CDF)** : This shows the percentage of data before and after a certain point.

6. Normal Distribution

It is known as the Gaussian distribution, is a probability distribution that is symmetric about the mean, showing that data near the mean are more frequent in occurrence than data far from the mean.

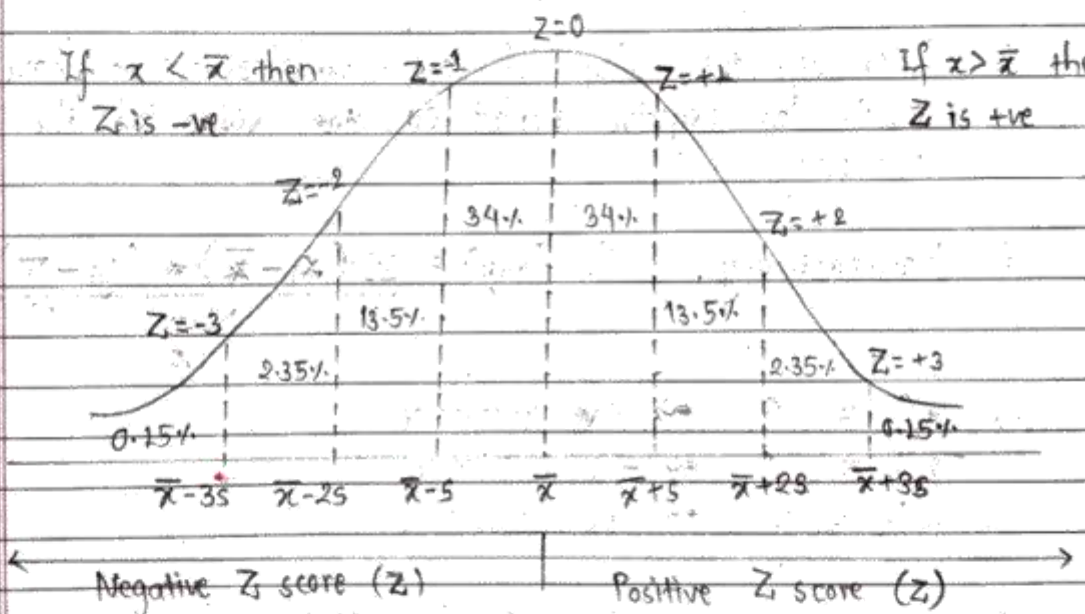
In graph form, normal distribution will appear as a bell curve.



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

Standard Normal Distribution

- The standard normal distribution, also known as the Z distribution or Z-score, is a special case of the normal distribution.
- mean (μ) of 0 and a standard deviation (σ) of 1.



In diagram: $\bar{x}$ = mean score
x = individual score.

7. Covariance and Correlation.

Covariance

- Covariance signifies the direction of the linear relationship between the two variables. By direction we mean if the variables are directly proportional or inversely proportional to each other.
- Increasing the value of one variable might have a positive or negative impact on the value of the other variable.
- Mathematically, $Cov(x,y) = \frac{\sum (x_i - \bar{x}) * (y_i - \bar{y})}{N}$
- $x\uparrow \Rightarrow y\uparrow$ ⇝ +ve covariance.
  $x\downarrow \Rightarrow y\uparrow$ ⇝ -ve covariance.
  $x\uparrow\downarrow \Rightarrow y$ ⇝ no covariance.
- The range of covariance is $-\infty$ to ∞.

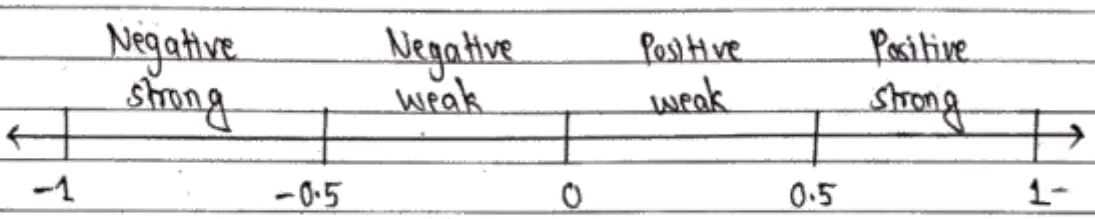
Correlation

- Correlation is a method of statistical evaluation used to study the strength of a relationship between two, numerically measured, continous variables.
- Mathematically, $Correlation = \frac{Cov(x,y)}{\sigma_x * \sigma_y}$

Where,

cov is covariance
σx is standard deviation of x
σy is standard deviation of y.

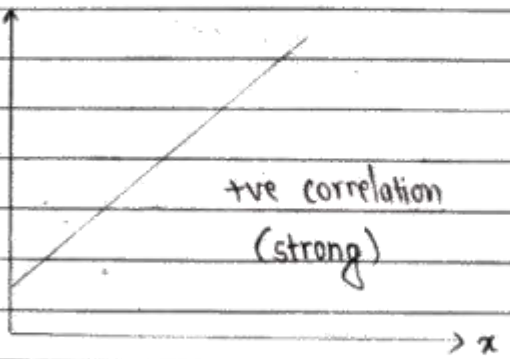
- The range of correlation is -1 to 1 which is converted from range of covariance for better analysis.



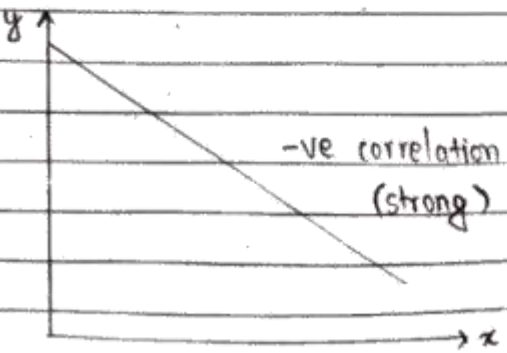
Parts of correlation

-

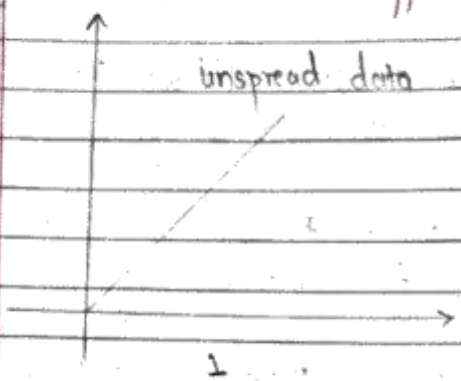
x	y
1	2
2	3
3	4
4	5
5	6

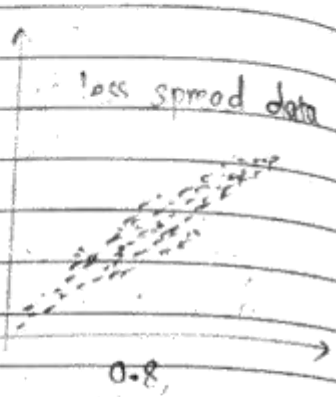


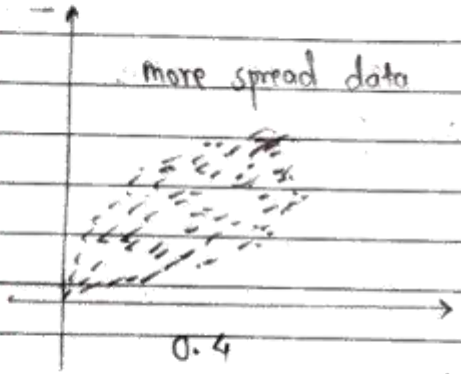
x	y
1	5
2	4
3	3
4	2
5	1

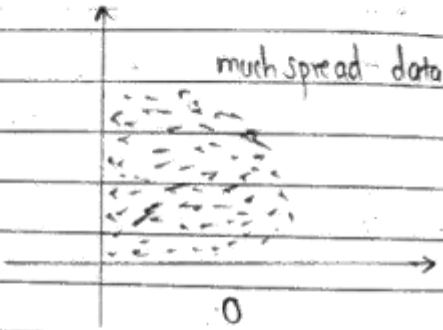


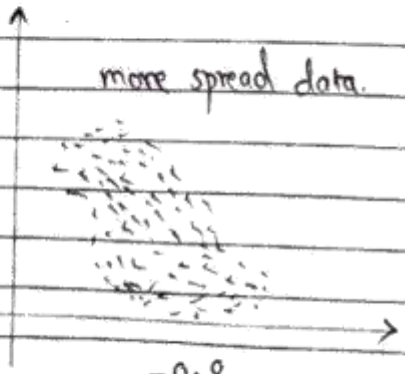
Pearson Correlation Coefficient:

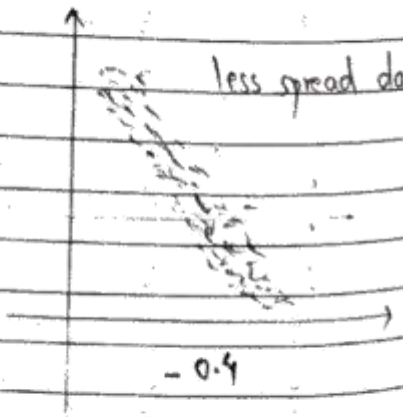


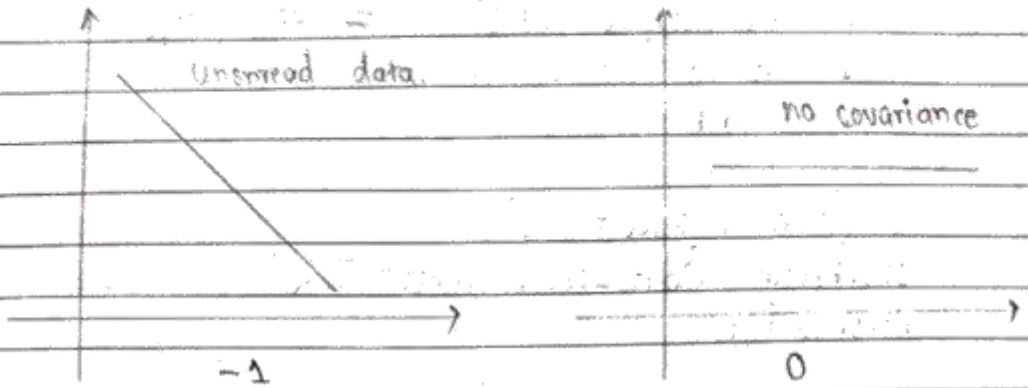












Working Practically in Jupyter:

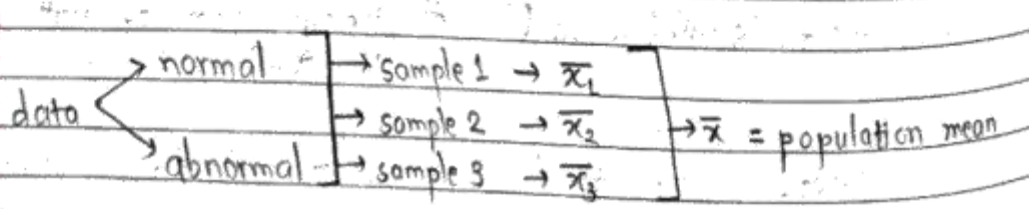
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

dataset = pd.read_csv("Students.csv")

dataset.head(3) # display first three head rows.

dataset.isnull().sum() # displays sum of null values
                       #in each columns.

dataset.info() # You can check data types using this.

data_corr = dataset.select_dtypes(["float64","int64"]).corr()
            # find correlation on selected datatypes.
data_corr # displays what data_corr variable stores.

data_cov = dataset.select_dtypes(["float64","int64"]).cov()
data_cov # For covariance
```

```
plt.figure(figsize=(4,3))    # adjust size
sns.heatmap(data_corr, annot=True)
plt.show()

plt.figure(figsize=(4,3))
sns.heatmap(data_cov, annot=True)
plt.show();
```

Correlation is more used than covariance in machine learning.

8. Central Limit Theorem

The central limit theorem (CLT) states that when plotting a sample distribution of means the means of the sample will be equal to the population mean and the sample distribution will approach normal distribution with variance equal to standard error.



There are few assumption behind the CLT:

- The sample data must be sampled and selected randomly from the population.
- There shouldn't be any multicollinearity in the sampled data which is one sample should not influence the other samples.

- The sample size should be no more than 10% of the population. Generally, sample size greater than 30 ($n > 30$) is considered good.

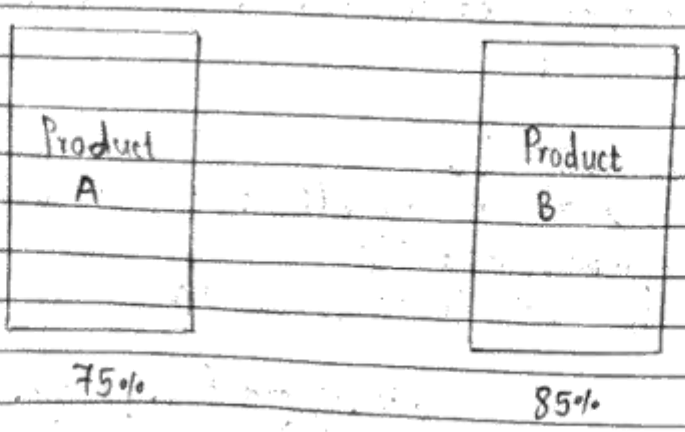
Working practically in Jupyter:

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

pop_data = [np.random.randint(10,100) for i in range
                                (10000)]
    # Creates a list of 10000 integers generated randomly.
pop_table = pd.DataFrame({"Pop_data": pop_data})
    # Creates a data frame with column Pop_data and
    # rows pop_data.

np.mean(pop_data) # Find mean of pop_data.

plt.figure(figsize=(5,5))
sns.kdeplot(x="Pop_data", data=pop_table)
plt.show();

sample_mean = []
for no_sample in range(60): #select 60 samples
    sample_data = []
    for data in range(500): # Select 500 data in
                            #each sample
        sample_data.append(np.random.choice(pop_data))
    sample_mean.append(np.mean(sample_data))
```

```
np.mean(sample_mean)

sample_M = pd.DataFrame({"Sample_Mean": sample_mean})

plt.figure(figsize=(5,5))
sns.kdeplot(x="Sample_Mean", data=sample_M)
plt.show();

# We find mean of pop-data and sample_mean is
# nearly equal which proves CLT.
```

9. Hypothesis Testing

- It is a part of statistical analysis, where we test the assumptions made regarding a population parameter.
- It is generally used when we were to compare a single group with an external standard and two or more groups with each other.



Termonology Used in Hypothesis Testing

- **Null Hypothesis :** is a statistical theory that suggests there is no statistical significance exists between the populations. It is denoted by H_0 and read as 'H-naught.

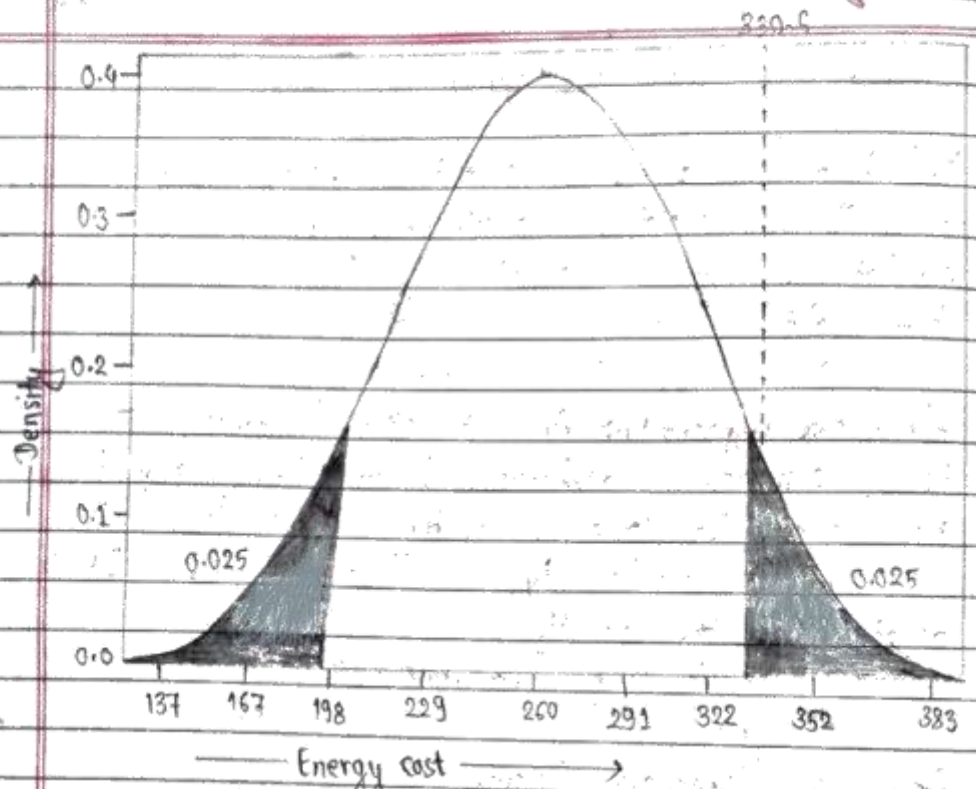
- **Alternative Hypothesis :** An alternative hypothesis suggests there is a significant difference between the population parameters. It could be greater or smaller. Basically, it is the contrast of the Null Hypothesis. It is denoted by H_a or H_1

Steps of Hypothesis Testing

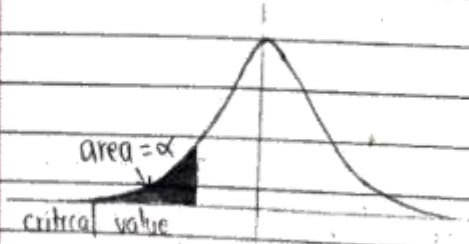
- State null (H_0) and alternative (H_1) hypothesis.
- Choose level of significance (α).
- Find critical values.
- Find test statistic
- Draw your conclusion.

- **State null (H_0) and alternative (H_1) hypothesis :**
  H_0 must always contains equality ($=$).
  H_1 always contain differences ($\neq$, $>$, $<$).

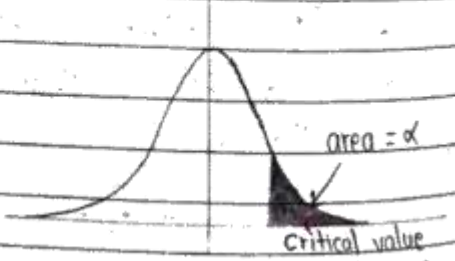
- **Choose level of significance :**
  It is a fixed probability of wrongly rejecting a True Null Hypothesis.
  It is denoted by α (alpha).



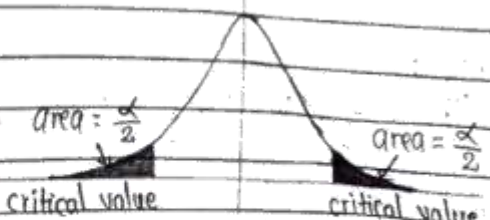
Visual representation of level of significance



left tailed (worse)



right tailed (better)



two tailed (non-directional)

Reject H_0 (H_0)

Do not reject H_0 (H_0)

- **Find critical values:**
  Using the test statistic find critical values like z value, t value, etc.

- **Find test statistic:**
  Find the required test like: Z-test, T-test, Chi-square test, etc. These test gives critical values.

Z test ⟷ population mean
Z test ⟷ population standard deviation → sample mean
Z test ⟷ sample ≥ 30

T test ⟷ population mean
T test ⟷ sample standard deviation → sample mean
T test ⟷ sample < 30

Chi-square test → checks goodness and relations between two variables (independance)

- **Draw your conclusion:**
  Conclude which one to be choosen either H_0 or H_a for better performance.

1. Z Test

↳ Data should be normally distributed

↳ Given:
- population mean
- population standard deviation
- sample mean
- no. of sample ≥ 30

Mathematically,

$$Z_{test} = \frac{\bar{x} - \mu}{\left(\frac{\sigma}{\sqrt{n}}\right)}$$

where, $\bar{x}$ = sample mean
μ = population mean
σ = population standard deviation
n = no. of sample

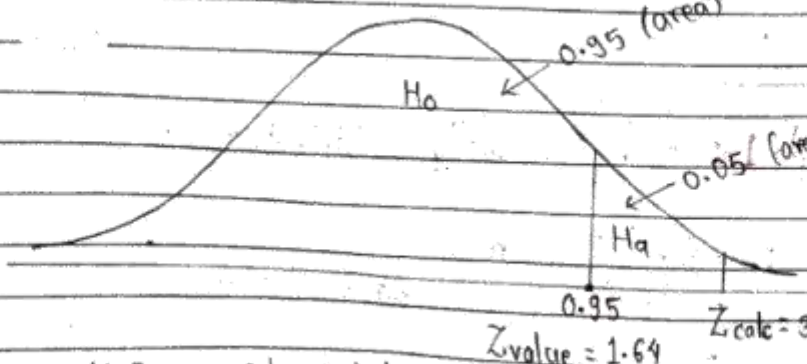
Example 1:

A teacher claims that the mean score of students in his class is greater than 82 with a standard deviation of 20. If a sample of 81 students was selected with a mean score of 90.

Solution:

H_o : $\mu \neq 82$ (opposing / null hypothesis)
H_a : $\mu > 82$ (supporting / alternative hypothesis)
Assume that $\alpha = 0.05$
Given, $\bar{x} = 90$
$\sigma = 20$
$n = 81$

Total area under curve is always 1 being a probability distribution



It is right tailed so, Zvalue should be of $(1-\alpha)$

Date
Page 39

Z_{value} = Sum of row and column z values of $(1-\alpha)$ area from same tailed z table (choose close value)

= 1.6 + 0.04 = 1.64

$$Z_{calc} = \frac{\bar{x} - \mu}{\left(\frac{\sigma}{\sqrt{n}}\right)} = \frac{90 - 82}{\left(\frac{20}{\sqrt{81}}\right)} = \frac{8 \times 9}{20} = 3.6$$

$\because Z_{calc} > Z_{value}$, the teacher was right.

Working practically in Jupyter:

```
import scipy.stats as st # library with constants
                         # stats has statistical analysis
import numpy as np

sample_mean = 90
population_mean = 82
population_std = 20
n = 81
alpha = 0.05  # Assume

z_calc = (sample_mean - population_mean) / (population_std /
                                            np.sqrt(n))
z_calc

z_table = st.norm.ppf(1-alpha) # norm contains functions
                               # related to normal distribution
z_table

if z_table < z_calc :
    print("Teacher is right")
else:
    print("Teacher is wrong")
```

Example 2:

Scenario:

Imagine you work for an e-commerce company and your team is responsible for analyzing customer purchase data. You want to determine whether a new website design has lead to a significant increase in the average purchase amount compared to the old design.

Data:

You have collected data from a random sample of 30 customers who made purchase on the old website design and 30 customers who made purchase on the new website design. You have the sample means, sample standard deviations, and sample sizes for both groups.

- Old design data = [45.2, 42.8, 38.9, 48.5, 41.0, 44.6, 40.5, 42.7, 39.8, 41.4, 44.3, 39.7, 42.1, 40.6, 43.0, 42.2, 41.5, 39.6, 44.0, 43.1, 38.7, 43.9, 42.0, 41.9, 42.8, 43.7, 41.3, 40.9, 42.5, 41.6]

- New design data = [48.5, 49.1, 50.2, 47.8, 48.7, 49.9, 48.0, 50.5, 49.8, 49.6, 48.2, 48.9, 49.7, 50.3, 49.4, 50.1, 48.6, 48.3, 49.0, 50.0, 48.4, 49.3, 49.5, 48.8, 50.6, 50.4, 48.1, 49.2, 50.7, 50.8]

- Population standard deviation = 2.5

Solution:

H_0 : new website = old website

H_a : new website > old website

Assuming no change in population mean i.e. $\mu_{new} = \mu_{old}$
Given, $\sigma = 2.5$
$n = 30$
We need, $\bar{x}_{new}$ and $\bar{x}_{old}$ → calculate from given samples
Now, assuming $\alpha = 0.05$:
Find z_{value} from z table, $z_{value} = 1.645$
And,

$$z_{calc} = \frac{(\bar{x}_{new} - \bar{x}_{old}) - (\mu_{new} - \mu_{old})}{\left(\frac{\sigma}{\sqrt{n}}\right)}$$

Since, $\mu_{new} = \mu_{old}$

$$\therefore z_{calc} = \frac{\bar{x}_{new} - \bar{x}_{old}}{\left(\frac{\sigma}{\sqrt{n}}\right)} = 16.110$$

Hence, you can draw your conclusion that H_a is right i.e. new website is better than old website.

Working practically in Jupyter :

```
import scipy.stats as st
import numpy as np

Old_design_data = np.array([<Old design data>])
New_design_data = np.array([<New design data>])
        # np.array() converts lists to array for fast
        # calculation

pop_std = 2.5
n_sp = len(New_design_data)   # 30
alpha = 0.05 #Assume
```

```
mean_new = np.mean (New_design_data)
mean_old = np.mean (Old_design_data)

z_cal = (mean_new - mean_old) / (pop_std / np.sqrt(n_sp))
z_cal

z_table = st.norm.ppf (1- alpha)
z_table

if z_calc > z_table :
    print (" New website is better than old.")
else :
    print (" New website is same as old.")
```

2. T Test

- Data should be normally distributed.
- Given :
  - population mean
  - sample mean
  - sample standard deviation
  - no. of sample < 30
- Mathematically,

$$t_{test} = \frac{\bar{x} - \mu}{\left(\frac{s}{\sqrt{n}}\right)}$$

Where, $\bar{x}$ = sample mean
μ = population mean

s = standard deviation of sample
n = no. of sample

Example 1:

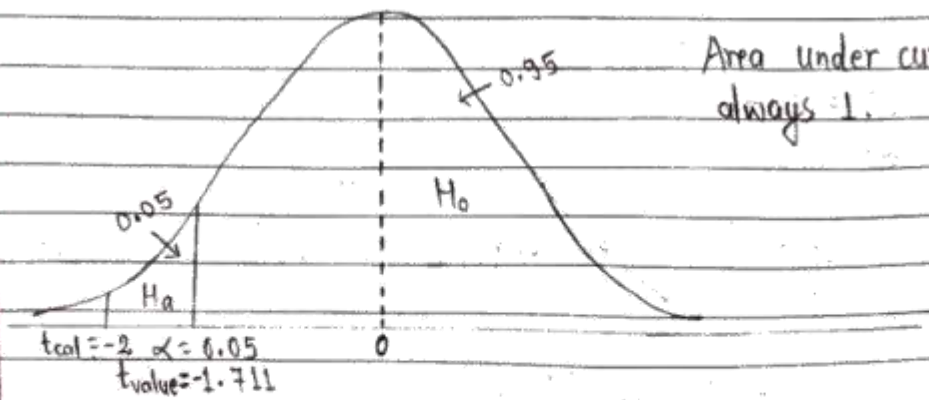
A manufacturer claims that the average weight of a bag of potato chips is 150 grams. A sample of 25 bags is taken and the average weight is found to be 148 grams, with a standard deviation of 5 grams. Test the manufacturer's claim using a one-tailed t-test with a significance level of 0.05.

Solution:

H_0: $\mu = 150$ gm (null hypothesis)
H_a: $\mu < 150$gm (alternative hypothesis)

Given,

$\alpha = 0.05$
$\mu = 150$
$\bar{x} = 148$
$s = 5$
$n = 25$



It is left tailed so, tvalue should be of α

degree of freedom, df = n-1 = 25-1 = 24.

t_{value} = the value that is present in df row and α column in t-table.
= -1.711 (-ve sign indicates it is left tailed)

$$t_{cal} = \frac{\bar{x} - \mu}{\left(\frac{s}{\sqrt{n}}\right)} = \frac{148 - 150}{\left(\frac{5}{\sqrt{25}}\right)} = \frac{148 - 150}{1} = -2$$

Hence, the claims of the company is wrong as we find $t_{cal} < t_{value}$.

Working practically in Jupyter:

```
import scipy.stats as st
import numpy as np

alpha = 0.05
pop_mean = 150
sam_mean = 148
sam_std = 5
n = 25

df = n -1
df

t_calc = (sam_mean - pop_mean) / (sam_std / np.sqrt(n))
t_calc

t_table = st.t.ppf(alpha, df)
t_table
```

if t_calc > t_table :
    print ("The claim of manufracturer is right")
else:
    print ("The claim of manufracturer is wrong")

Example 2:

A company wants to test whether there is a difference in productivity between two teams. They randomly select 20 employees from each team and record their productivity scores. The mean productivity for Team A is 80 with a standard deviation of 5, while the mean productivity score for Team B is 75 with a standard deviation of 6. Test at a 5% level of significance whether there is a difference productivity between two teams.

Solution:

H_0 : Productivity of A − Productivity of B = 0
H_a : Productivity of A − Productivity of B ≠ 0 (two tailed)
Given,

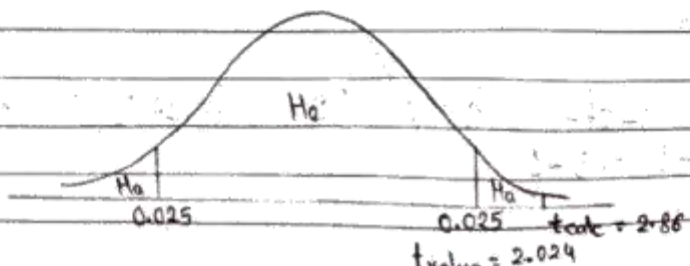
$2\alpha = 5\% = 0.05 \Rightarrow \alpha = 0.025$ [∵ two tailed]

$n_A = n_B = 20$

$\overline{x}_A = 80$ and $S_A = 5$

$\overline{x}_B = 75$ and $S_B = 6$

Assume that $\mu_A = \mu_B$.



degree of freedom, $df = n_A - 1 + n_B - 1$

$= 20 - 1 + 20 - 1$

$= 38$

$t_{value} = -2.024$ [From t-table]

$$t_{calc} = \frac{(\bar{x}_A - \bar{x}_B) - (\mu_A - \mu_B)}{\sqrt{\frac{S_A^2}{n_A} + \frac{S_B^2}{n_B}}}$$

Since, $\mu_A = \mu_B$

$$\therefore t_{calc} = \frac{\bar{x}_A - \bar{x}_B}{\sqrt{\frac{S_A^2}{n_A} + \frac{S_B^2}{n_B}}} = 2.86$$

Since, $t_{calc} > t_{value}$ there is some difference between productivity of A and productivity of B.

Working practically in Jupyter:

```
import numpy as np
from scipy.stats import t

level_of_significance = 5/100   # 5%
nA = nB = 20
mean_A = 80
std_A = 5
mean_B = 75
std_B = 6

alpha = level_of_significance / 2   # It is two tailed
df = nA - 1 + nB - 1
```

```
t_table = t.ppf (alpha, df)   # Testing for left tail
t_calc = (mean_A - mean_B) / (np.sqrt ((np.square (std_A)
          /nA) + (np.square (std_B)/nB))
print ("t_table =", t_table, "and t_calc =", t_calc)

if t_calc > t_table :
    print ("Productivity of A != Productivity of B")
else :
    print ("Productivity of A = Productivity of B")
```

Example 3:

A company wants to test whether a new training program improves the typing speed of its employees. The typing speed of 20 employees was recorded before and after the training program. The data is given below. Test at a 5% level of significance whether the training program has an effect on the typing speed of the employees.

- Before : 50, 60, 45, 65, 55, 70, 40, 75, 80, 65, 70, 60, 50, 55, 45, 75, 60, 50, 65, 70

- After : 60, 70, 55, 75, 65, 80, 50, 85, 90, 70, 75, 65, 55, 60, 50, 80, 65, 55, 70, 75

Solution :

H_0 : Speed before training − Speed after training = 0

H_a : Speed before training − Speed after training ≠ 0

Given,

$\alpha = 5\% = 0.05 \Rightarrow \frac{\alpha}{2} = 0.025$ [two tailed test]

$n = 20$

degree of freedom, $df = n - 1 = 20 - 1 = 19$

Assume that $\mu_{after} = \mu_{before}$ [$\because$ not given]

Calculate $\bar{x}_{before}$, $\bar{x}_{after}$, s_{before} and s_{after} from the given sample data.

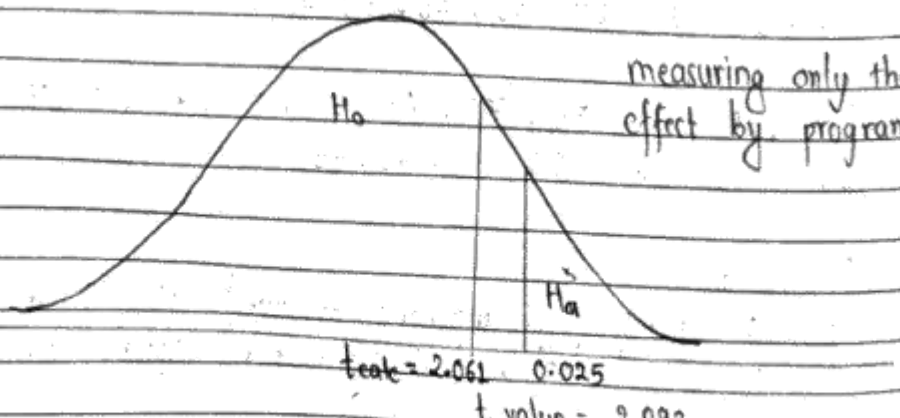
t-value = 2.093 [from t-table]

$$t\text{-calc} = \frac{(\bar{x}_{after} - \bar{x}_{before}) - (\mu_{after} - \mu_{before})}{\sqrt{\frac{s_{after}^2}{n_{after}} + \frac{s_{before}^2}{n_{before}}}}$$

$\because \mu_{after} = \mu_{before}$ and $n_{after} = n_{before} = n$

$$\therefore t\text{-calc} = \frac{\bar{x}_{after} - \bar{x}_{before}}{\sqrt{\frac{s_{after}^2 + s_{before}^2}{n}}} = 2.061$$

Clearly, $2.061 < 2.093$ i.e. t-calc < t-value so, the training program doesn't have any effect on typing speed of employees.



measuring only the +ve effect by program.

two tailed but analyzing only right tail.

Working practically in Jupyter :

```
import numpy as np
from scipy.stats import t

before = np.array([<before data>])
after = np.array([<after data>])
level_of_significance = 5 / 100   # 5%
n = 20

alpha = level_of_significance / 2   # It is two tailed
df = n-1

mean_before = np.mean(before)
mean_after = np.mean(after)
std_before = np.std(before)
std_after = np.std(after)

t_table = t.ppf(1-alpha, df)  # Testing for right tail
t_calc = (mean_after - mean_before) / np.sqrt((np.
        square(std_after) + np.square(std_before))/n)
print(f"t_table = {t_table} and t_calc = {t_cal}")

if t_calc > t_table:
    print("The training program is effective")
else:
    print("The training program is not effective")
```

3. Chi Square Test

- Applicable in two types of situations :
  i. Goodness
  ii. Independance

- Mathematically, $$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

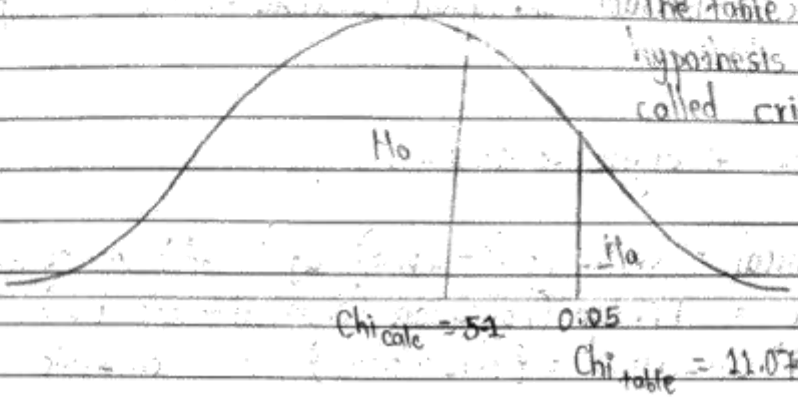
  where, χ^2 = chi square
  O_i = observed value
  E_i = expected value

Example 1 :
A fair dice is rolled 120 times and the following results are obtained :
Face 1 : 22 times
Face 2 : 17 times
Face 3 : 20 times
Face 4 : 26 times
Face 5 : 22 times
Face 6 : 13 times
Test at 5% level of significance whether the dice is fair.

Solution :
H_0 : dice is fair
H_a : dice is not fair

In dice, number of faces, $n = 6$
Degree of freedom, $df = n - 1 = 6 - 1 = 5$

level of significance, $\alpha = 5\% = 0.05$



The expected value for each face when a dice is rolled 120 times is 20 for each.
The observed values are given above.

Chi_{table} = the value that is present in df row and α column in chi square table.
= 11.07

$$Chi_{calc} = \sum \frac{(O_i - E_i)^2}{E_i} = 5.1$$

Clearly $Chi_{calc} < Chi_{table}$ so, the dice is fair i.e. Ho is right.

Working practically in Jupyter

```
import numpy as np
from scipy.stats import chi2

observed = np.array([22, 17, 20, 26, 22, 18])
expected = np.array([20, 20, 20, 20, 20, 20])
```

```
alpha = 0.05  # Given level of significance is 5%
n=6  # Total number of possibilities when a dice is
     # rolled

df = n-1  # degree of freedom

chi_critical = chi2.ppf (1-alpha), df)  # Right tailed
chi_cale = np.sum (np.square (observed - expected) / expected)
print (f" chi_critical = {chi_critical} and chi_calc = {chi_calc}")

if chi_cale > chi_critical :
    print ("Dice is not fair")
else :
    print ("Dice is fair")
```

Example 2.

A study was conducted to investigate whether there is a relationship between gender and the perferred genre of music. A sample of 235 people was selected, and the data collected is shown below. Test at a 5% level of significance whether there is significant association between gender and music preference.

	Pop	Hip Hop	Classical	Rock
Male	40	45	25	10
Female	35	30	20	30

Solution:

H_0 : No association

H_a : Has association

Given:
no. of sample, n = 235
level of significance, α = 5% = 0.05
no. of rows, rows = 2
no. of columns, columns = 4
Now,

$$df = (\text{rows} - 1) * (\text{columns} - 1)$$
$$= (2-1) * (4-1)$$
$$= 3$$

So,

$$\text{chi}_{\text{critical}} = 7.815 \quad [\text{From chi-square table}]$$

We are given the observed values in the given table. To find the expected value for each observed value proceed as follows:

i.) find the sum of each column and row.

ii.) expected value $(i, j) = \frac{\sum R_i \times \sum C_j}{\text{No. of sample}}$

So,

$\sum R_1 = 120$ and $\sum R_2 = 115$

$\sum C_1 = 75$, $\sum C_2 = 75$, $\sum C_3 = 45$ and $\sum C_4 = 40$

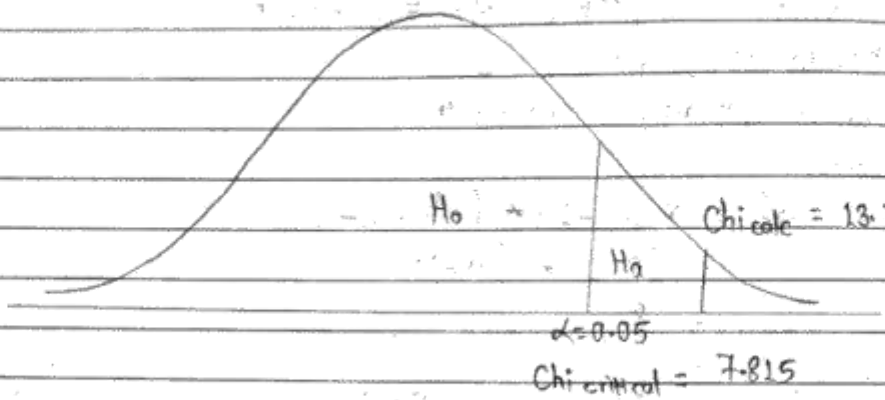
We've,
observed values = [40, 45, 25, 10, 35, 30, 20, 30]
for expected values:

expected value (1,1) = $\frac{\sum R_1 \times \sum C_1}{n} = 38.29$

expected value (1,2) = $\frac{\sum R_1 \times \sum C_2}{n} = 38.29$

Similarly, all the expected values can be calculated.
Finally,
expected values = [38.29, 38.29, 22.97, 20.42, 36.70, 36.70, 22.02, 19.57]

$$chi_{calc} = \sum \frac{(O_i - E_i)^2}{E_i} = 13.78$$



Clearly $chi_{calc} > chi_{critical}$ so, there is association between gender and music preference.

Working practically in Jupyter:

```
import numpy as np
from scipy.stats import chi2

alpha = 0.05 # Given level of significance is 5%
n = 235 # Sample of 235 people was selected

rows = 2 # from table
columns = 4

row1 = np.array([40, 45, 25, 10])
row2 = np.array([35, 30, 20, 30])

df = (rows-1) * (columns-1) # Calculate degree of freedom
df
```

```
chi_critical = chi2.ppf (alpha, df)
print ("chi-critical =", chi_critical)

sum_row1 = np.sum (row1)
sum_row2 = np.sum (row2)
sum_rows = np.array ([sum_row1, sum_row2]) # Create an
# array of sum of rows.
sum_rows

sum_columns = row1 + row2  # Find sum of all columns at
once in an array.
sum_columns

# Find expected values
expected = []
for i in sum_rows :
    for j in sum_columns:
        expected.append (i*j/n)
expected

# Get observed values
observed = np.append (row1, row2)
observed

chi_calc = np.sum (np.square (observed - expected) / expected)

if chi_calc > chi_critical :
   print ("There is association")
else:
    print ("There is no association")
```

3. Machine Learning

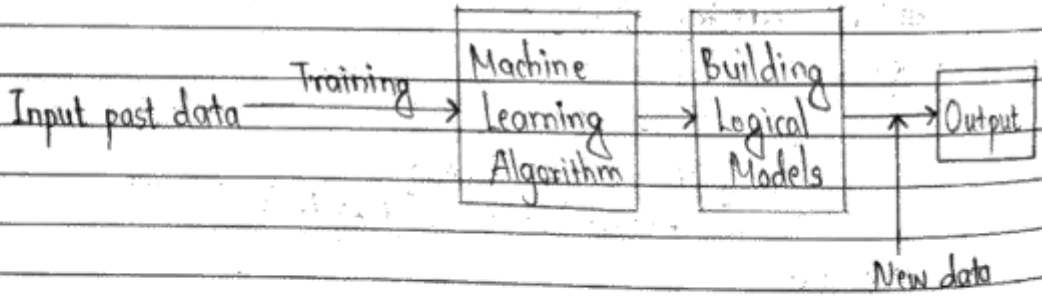
Date
Page 56

1. Introduction

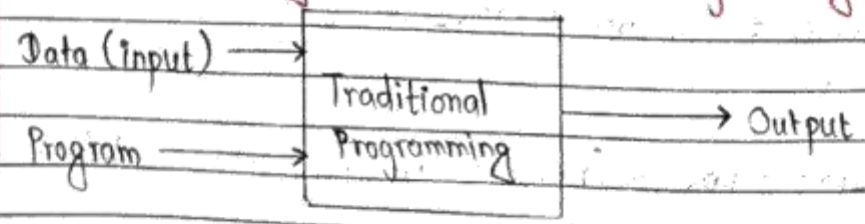
Machine Learning is the science of getting computers to learn and act like humans do, and improve their learning over time in autonomous fashion, by feeding them data and information in the form of observations and real world interactions.

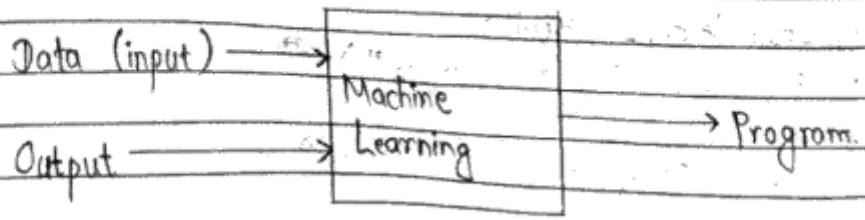
How Does Machine Learning Work ?

A machine learning system learns from historical data, builds the prediction models and whenever it receives new data, predicts the output for it.



Machine Learning V/s. Traditional Programming





Date
Page 57

Classification of Machine Learning

1. Supervised Learning → future predictions / recommendation sys
2. Unsupervised Learning → tasks related to groupism / grouping
3. Reinforcement Learning → used in game automation

Advantages of Machine Learning

- Easily identifies trends and patterns.
- No human intervention needed (automation)
- Continues improvement.
- Handling multi-dimensional and multi-variety data
- Wide applications

Disadvantages of Machine Learning

- Data acquisition
- Time and resources
- Interpretation of results
- High error-susceptibility

Uses of Machine Learning

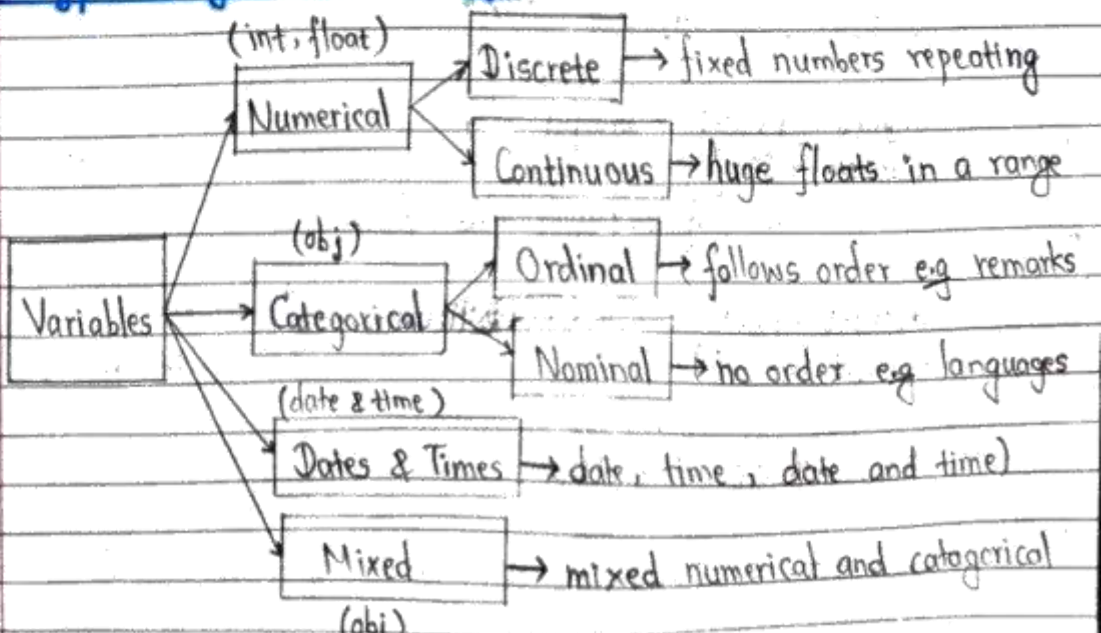
- Automatic language translation
- Medical diagnosis
- Stock market trading
- Online fraud detection
- Virtual personal assistant
- Email span and malware filtering
- Self driving cars
- Product recommendations
- Traffic prediction
- Speech recognition
- Image recognition

2. Road Map

- Programming language : Python (recommended), R
  - pandas → for data analysis
  - numpy → solve numerical problems
  - matplotlib → data analysis in graphical form
  - seaborn → data analysis in graphical form.
  - scipy → provides constants
  - scikit learn → machine learning models
  - tensorflow → deep learning
  - nltk → text processing
  - open cv → image processing
- Exploratory data analysis
- Feature engineering
  - Exploratory data analysis
  - Handling missing value
  - Handling outliers
  - Categorical encoding
  - Normalizing and standardization
- Feature selection
  - Correlation
  - Forward elimination
  - Backward elimination
  - Univariate selection
  - Random forest importance
  - Feature selection with decision trees

- Machine learning algorithms → regression and classification, clustring
- Linear, Logistic Regression, Decision Tree, Random Forest, K means
- Grid Search, Randomised Search, Hyperopot, Genetics algorithms.
- Dockers and Kubernetes
- Model deployments
- End to End ML Projects

3. Types of Variables



4. Data Cleaning

Data cleaning is the process of preparing data for analysis / ML / DL by removing and modifying data that is incorrect, incomplete, irrelevant, duplicated, or improperly formatted.

Steps of Data Cleaning

1. Handling missing data
2. Outlier detection and handling
3. Data scaling and transformation
4. Encoding categorical variables
5. Handling duplicates
6. Dealing with inconsistent data

1. Handling Missing Data

Finding out missing values:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

dataset = pd.read_csv(r".\Students.csv")      #r means raw string
dataset.head(8)                               #. means current path

dataset.shape  # Displays (rows, columns)

dataset.shape[0] # Displays no. of rows, 1 for columns
```

```
dataset.isnull()  # Represent True if null value otherwise
# false in dataset

dataset.isnull().sum() # Provides total number of null
# values in each column.

dataset.isnull().sum().sum() # Provides total number of
# null values in dataset

# Similarly, notnull() can be used for non-null values
dataset.notnull().sum().sum()

(dataset.isnull().sum()/dataset.shape[0])*100 # Shows
# total null values in each column in percentage.

(dataset.isnull().sum().sum()/(dataset.shape[0]*dataset.shape
[1]))*100  # Shows total percentage of null value.

# for making a graphical representation
sns.heatmap(dataset.isnull())
plt.show()
```

Dropping missing values :

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

dataset = pd.read_csv("Students.csv")
dataset.head(9)
```

```
dataset.shape  # Provides total number of rows and columns

dataset.isnull().sum()  # Provides total number of null
# values in each column.

# lets plot the graphical visual
sns.heatmap(dataset.isnull())
plt.show()

# Droping a column with maximum null values
dataset.drop(columns=["Gender"], inplace=True)

# Droping all the rows with null values
dataset.dropna(inplace=True)

dataset.shape  # Provides remaining rows and columns

dataset.isnull().sum().sum()  # It must be zero

# lets plot a graph
sns.heatmap(dataset.isnull())
plt.show()
```

Imputing category data:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

dataset = pd.read_csv("Students.csv")
```

```
dataset.head(5)

dataset.isnull().sum()

dataset.fillna(5)  # fills all the missing values with 5
# Remember that we have many types of data with
# different data types so, filling same thing to all is
# not a convinent way.
# Category data can be filled using mode, backward
# filling or forward filling

dataset.info()  # Shows information related to dataset.

dataset.fillna(method="bfill")  # Backward filling

dataset.fillna(method="ffill")  # Forward filling
# You can also choose axis, axis=0 is default.

# Filling mode in a particular column
dataset["Age"].fillna(dataset["Age"].mode()[0])

# Filling mode in all columns having categorical data.
for i in dataset.select_dtypes(include="object").columns:
    dataset[i].fillna(dataset[i].mode()[0])
# Use inplace=True if you want a permanent change.

# Lets plot a graph
sns.heatmap(dataset.isnull())
plt.show()
```

Imputing missing values using scikit-learn :

```
import pandas as pd

dataset = pd.read_csv("Students.csv")
dataset.head(5)

dataset.isnull().sum()

dataset.info()

dataset.select_dtypes(include="float64").columns
# Returns all the columns having float64 datatype.

columns = dataset.select_dtypes(include="float64").columns
# Store the columns in the variable as a list.

# Now imputing missing values using scikit-learn
from sklearn.impute import SimpleImputer

si = SimpleImputer(strategy="mean") # Create an object of class
si.fit_transform(dataset[columns]) # This will return the
# columns as in the form of array after filling null values

array = si.fit_transform(dataset[columns])
new_dataset = pd.DataFrame(array, columns=columns)
new_dataset # Transformed array to dataset.

new_dataset.isnull().sum() # No null values are present
# All the null values have been filled by means.
```

2. Outlier Detection and Handling

Outlier Detection :

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

dataset = pd.read_csv ("Students.csv")
dataset.head ()

dataset.info () # Provides info about dataset

dataset.describe () # Describe dataset statistics

#Visual using boxplot
sns.boxplot (x="Age", data = dataset )
plt.show()

# We find no outliers in the data
# So lets create some outliers in the Age column by ourself
# for analysis by filling null values.
import random
dataset ["Age"].fillna(random.choice ([i for i in range (0,5)]
    + [i for i in range (80,95)]), inplace=True)

sns.boxplot (x = "Age", data = dataset)
plt.show ()
# Now we can see outliers clearly in the boxplot
```

Outlier Handling:

Outlier Handling by Using Direct Method :

```
min_range = dataset["Age"].mean() - (3 * dataset["Age"].std())
        # min_range = mean - 3std
max_range = dataset["Age"].mean() + (3* dataset["Age"].std())
        #max_range = mean + 3std.
min_range , max_range
```

```
# Create a new dataset by removing outliers.
new_dataset = dataset[(dataset["Age"] <= max_range) &
                      (dataset["Age"] >= min_range)]
new_dataset.head(3)
```

```
sns.boxplot(x="Age", data = new_dataset)
plt.show()
```

"If some outliers are still remaining, avoid these because it can lead to huge data losses."

Outlier Removal Using IQR :

```
q1 = dataset["Age"].quantile(0.25) # First quantile
q2 = dataset["Age"].quantile(0.75) # Third quantile
q1, q3
```

```
iqr = q3 - q1  # Calculate IQR
iqr
```

```
# Find range
min_range = q1 - (1.5 * iqr)
max_range = q3 + (1.5 * iqr)
min_range , max_range

# Remove the outliers
new_dataset = dataset [(dataset ["Age"] <= max_range) &
                       (dataset ["Age"] >= min_rage)]

sns.boxplot (x = "Age" , data = new_dataset)
plt.show()
```

"If some outliers are still remaining, avoid them because removing can lead to huge data losses"

Outlier Removal by Using Z Score :

- The Z-score method has same functionality as direct method.
- Mathematically,

$$\text{Z-score} = \frac{x - \mu}{\sigma}$$

where, μ = mean
σ = standard deviation

- Zrange = $\mu - 3\sigma$ to $\mu + 3\sigma$ i.e. -3 to +3.
- We need to select data having z-score within Zrange.

```
# Calculate z-score
z-score = (dataset["Age"] - dataset["Age"].mean()) /
          dataset["Age"].std()
z-score

dataset["Z Score"] = z-score #Adding a column to dataset

new-dataset = dataset[(dataset["Z Score"] < 3) &
              (dataset["Z Score"] > -3)] # Removes outliner
new-dataset.head(4)

sns.boxplot(x="Age", data = new-dataset)
plt.show()
```

3 Data Scaling and Transformation

Feature Scaling (Standardization)

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

dataset = pd.read-csv("Students.csv")
dataset.head(3)

dataset.isnull().sum()

dataset["Age"].fillna(dataset["Age"].mean(), inplace=True)
# fill null values
```

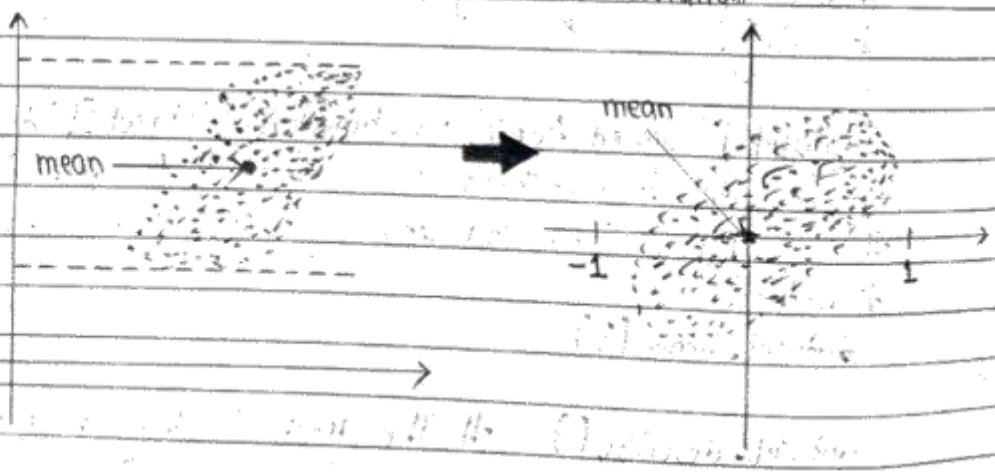
```
# Visualization.
sns.distplot(dataset["Age"])
plt.show()

dataset.describe  # Describe statistics of dataset

# Feature scaling using sklearn
from sklearn.preprocessing import StandardScalar

ss = StandardScalar()
ss.fit(dataset[["Age"]])  # Works upon two dimension
# dataset
# The fit(data) is used to compute the mean and std
# dev for a given feature to be used further for
# scaling.

dataset["Scaled Age"] =ss.transform(dataset[["Age"]])
# The transform(data) method is used to perform scaling
# using mean and std dev calculated using the .fit()
# method.
dataset.head(5)

dataset.describe()  # The mean is very near to zero

# For visualization
plt.figure(figsize=(10,7))
# Previous data graph
plt.subplot(1,2,1)
plt.title("Before")
sns.distplot(dataset["Age"])
# Scaled data graph
```

```
plt.subplot(1, 2, 2)
plt.title("After")
sns.distplot(dataset["Scaled Age"])
#Display plot
plt.show()
```

- **Standardization:** It is a very effective technique which re-scales a feature value so that it has distribution with 0 mean value and variance equals to 1.

- Mathematically,

$$X_{new} = \frac{X_i - X_{mean}}{\text{Standard deviation}}$$



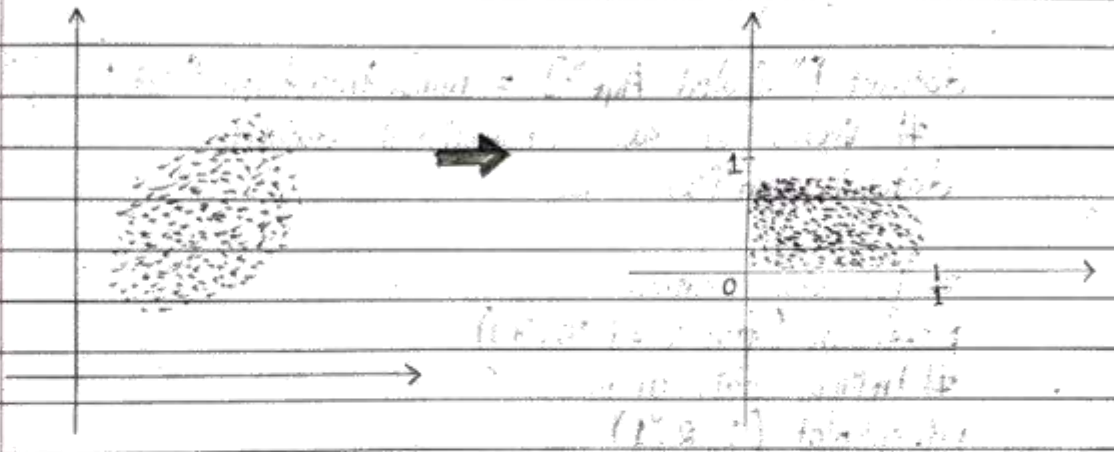
Feature Scaling (Normalization)

- **Normalization:** It is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1.

Date
Page 71

- It is also known as Min-Max scaling.
- Mathematically,

$$X_{new} = \frac{X_i - \min(X)}{\max(X) - \min(X)}$$




```
dataset = pd.read_csv("Students.csv")
dataset.head(3)

dataset.isnull().sum()

dataset["Age"].fillna(dataset["Age"].mean(), inplace = True)
# Filling up null values

# See graphical nature
sns.distplot(dataset["Age"])
plt.show()

# Feature scaling (normalization)
from sklearn.preprocessing import MinMaxScaler
```

```
mms = MinMaxScaler()
mms.fit(dataset[["Age"]]) # Works upon two dimension dataset.
# The fit(data) method is used to compute the mean
# and std dev for a given feature to be used further
# for scaling.

dataset["Scaled Age"] = mms.transform(dataset[["Age"]])
  # Works on two dimensional dataset
dataset.head(3)

# For visualization
plt.figure(figsize=(10,7))
# Previous data graph
plt.subplot(1,2,1)
plt.title("Before")
sns.distplot(dataset["Age"])
# Scaled data graph
plt.subplot(1,2,2)
plt.title("After")
sns.distplot(dataset["Scaled Age"])
# Show graph
plt.show()
```

4. Encoding Categorical Variables

One Hot Encoding and Dummy Variables

- It is used when you have limited amount of categorical data.

```
import pandas as pd

dataset = pd.read_csv("Students.csv")
dataset.head()

dataset.isnull().sum() # Check for missing values

# Fill all the missing values with mode of the column
# which we want to encode.
dataset["Gender"].fillna(dataset["Gender"].mode()[0],
                         inplace = True)
dataset["Enrollment Status"].fillna(dataset["Enrollment
          Status"].mode()[0], inplace = True)
# Now the data is ready for encoding
```

Using Pandas : get_dummies()

```
en_data = dataset[["Gender", "Enrollment Status"]]
# Getting all the columns that needs to encode.

encoded_data = pd.get_dummies(en_data)
# Encoding our data (one hot encoding)
encoded_data

encoded_data.info() # After encoding we have our data
# in boolean form

# But we need data in numerical form
encoded_data.apply(lambda x : x.astype(int)) # Convert
# boolean data to numerical form.
```

# Now convert it in Data frame
numeric_data = encoded_data.apply(lambda x: x.astype(int))
pd.DataFrame(numeric_data, columns = numeric_data.columns)

Using scikit-learn : OneHotEncoder

from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder()
ohe.fit_transform(en_data) # Provides an output in form of sparse matrix
Sparse matrix is a matrix in which elements are filled
with 0 and 1.
The fit_transform() method does both fits and
transform.

# To get the data in form of array
ohe.fit_transform(en_data).toarray()

# But we need it in the form of data frame
ar = ohe.fit_transform(en_data).toarray()
pd.DataFrame(ar, columns = encoded_data.columns)

# If your data is too large and want to drop the first column you can proceed as
ohe = OneHotEncoder(drop = "first")
ar = ohe.fit_transform(en_data).toarray()
pd.DataFrame(ar, columns = [encoded_data.columns[1],
encoded_data.columns[3]])
Because first columns of both categorical data was
removed i.e. columns[0] and columns[2] are removed

Label Encoding

↳ It is performed on nominal data (data having no order sequence i.e. has no connections)

```
import pandas as pd

df = pd.DataFrame ({"Name" : ["Mukesh", "Nikesh",
                    "Ashok", "Sandip", "Subha"]})
df

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder ()
le. fit_transform (df["Name"]) # The fit_transform ()
# method does both fits and transform.

# If we need to add encoded data column to our
# Data Frame
df["Encoded Name"] = le.fit_transform (df["Name"])
df
```

Take some real world example and apply concept of label encoding

```
dataset = pd. read_csv ("Students.csv")
dataset.head (3)

label_enc = LabelEncoder ()
label_enc = fit (dataset["Faculty"])
# The fit (data) method is used to compute the
```

mean and std dev for a given feature to be
used further for scaling.

label_enc.transform(dataset["Faculty"])
# The transform(data) method is used to perform
# scaling using mean and std dev calculated by using
# the .fit() method.

If you want to replace the data with encoded data
dataset["Faculty"] = label_enc.transform(dataset["Faculty"])
dataset

Ordinal Encoding

- It is performed on ordinal data (data having order sequence i.e. has connections)

With Scikit-learn :

```
import pandas as pd

df = pd.DataFrame({"Size": ["s", "l", "xl", "m", "l",
                            "s", "xl", "m", "s", "xl", "l"]})
df.head(5)

data_order = [["s", "m", "l", "xl"]] # Two dimensional
# list for order of data.

from sklearn.preprocessing import OrdinalEncoder
```

```
oe = OrdinalEncoder(categories = data_order)
oe.fit(df[["Size"]])

oe.transform(df[["Size"]])

# If you want to add encoded data to Data Frame
df["Encoded With Scikit-learn"] = oe.transform(df
                                  [["Size"]])
df
```

With Map Function:

```
data_order = {"s":0, "m":1, "l":2, "xl":3}
# You can decide the numbers accordingly

df["Size"].map(data_order) # Ordinal encoding using
# map()

# If you want to save the encoded data to Data Frame
df["Encoded With Map function"] = df["Size"].map(
                                  data_order)
df
```

Performing ordinal encoding in large dataset:-

```
dataset = pd.read_csv("Students.csv")
dataset.head(4)

dataset["Level"].isnull().sum() # Checking for null
# values in Level column
```

```
dataset["Level"].fillna(dataset["Level"].mode()[0],
                        inplace=True) # filling up null values

dataset["Level"].unique() # Returns all the unique
# values present in the column

encode_data_order = [["Graduate", "Post Graduate", "Phd"]]
# order encoding

from sklearn.preprocessing import OrdinalOrder

ordinal_encoder = OrdinalEncoder(categories=encode_data_order)
dataset["Level"] = ordinal_encoder.fit_transform(dataset[[
                                "Level"]])
dataset.head(5)
```

5. Handling Duplicates

- Content is called duplicate if two rows are exactly same.
- Having same data in some columns doesn't mean it is duplicate.

```
import pandas as pd

data = {
    "Name": ["Mukesh", "Nikesh", "Binod", "Ashok", "Subha",
             "Nikesh", "Subham"],
    "Physics": [56, 67, 87, 45, 33, 67, 34],
```

```
    "Chemistry" : [56, 64, 34, 56, 45, 64, 89],
    "Biology" : [64, 67, 76, 45, 56, 67, 32],
}
df = pd.DataFrame(data)
df
```

```
df.duplicated() # Returns the duplicated rows

df["Duplicated"] = df.duplicated() # Adding duplicated
# data in data frame
df

# Remove the duplicated column which we have added
df.drop("Duplicated", axis=1, inplace=True)
df

df.drop_duplicates(keep="first", inplace=True)
df
```

6. Dealing With Inconsistent Data

Replace and Data Type Change

```
import pandas as pd

dataset = pd.read_csv("Students.csv")
dataset.head(3)

dataset.info() # Get info about dataset
```

```
dataset.isnull().sum() # Get the null values

# Fill null values in Phone Number column.
dataset["Phone Number"].fillna(dataset["Phone Number"]
        .mode()[0], inplace=True)

dataset["Phone Number"].value_counts() # Get count
# for each unique value present in dataset column

# Lets convert the data into the numerical data i.e.
# remove - sign
dataset["Phone Number"] = dataset["Phone Number"].str
        .replace("-","")
dataset.head(5)

# Convert data into numerical datatype
dataset["Phone Number"] = dataset["Phone Number"]
                .astype("int64")
dataset.head(5)

dataset.info() # You will see the change in datatype
```

5. Function Transformation

- Transformation of data by using mathematical functions:
  - → If you have non-normal distribution data
  - → If your data pattern is very high
  - → Want to change the units of data

Pros : It can be used multiple times on a same data to meet our need like getting normally distributed data and wide range of using functions as we can use any type of function.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

dataset = pd.read_csv("Students.csv")
dataset.head(9)

dataset.isnull().sum()

# Lets drop all the rows with null values in the age
# column
dataset["Age"].dropna(axis=0, inplace=True)

sns.distplot(dataset["Age"])
plt.show()

# The graph is not normally distributed so lets try to
# make it a distributed graph
# First lets check by removing outliers
q1 = dataset["Age"].quantile(0.25)
q3 = dataset["Age"].quantile(0.75)
iqr = q3 - q1
min_range = q1 - (1.5 * iqr)
max_range = q3 + (1.5 * iqr)
min_range, max_range
```

```
dataset = dataset[(dataset["Age"] >= min_range) &
                  (dataset["Age"] <= max_range)]
dataset.head(3)

sns.distplot(dataset["Age"])
plt.show()
"""No much progress found"""

# So lets try by us using function transformer
from sklearn.preprocessing import FunctionTransformer
import numpy as np

ft = FunctionTransformer(func=np.log1p) # Natural logarithm
# of 1+x , element wise

ft.fit(dataset["Age"])

dataset["Transformed Age"] = ft.transform(dataset["Age"])
dataset.head(4)

# for visualization
plt.figure(figsize=(10,7))
# Previous data graph
plt.subplot(1,2,1)
plt.title("Before")
sns.distplot(dataset["Age"])
# Scaled data graph
plt.subplot(1,2,2)
plt.title("After")
sns.distplot(dataset["Transformed Age"])
plt.show()
```

```
# You can also use your own function like this
fun_transformer = FunctionTransformer (fun = lambda x:
                                       x** 2)
dataset [" My Transformed Age" ] = fun_transformer. fit_
                transform (dataset ["Age"])
dataset. head (3)

plt. subplot (1, 2, 1)
plt. title ("Before")
sns. distplot (dataset ["Age"])
# Scaled data graph
plt. subplot (1, 2, 2)
plt. title ("After")
sns. distplot (dataset [" My Transformed Age"])
plt. show ()
```

6. Feature Selection Technique

- Select the feature which are important
- Drop the features which are not important

1. Forward Elimination

1. Begin with an empty model (no features)
2. Add features iteratively. In each iteration, add one feature that improves the model the most.

iii. Evaluate the model's performance using a choosen metric (accuracy, AUC, etc.)

iv. Add the feature that results in the best improvement to the model. Repeat the process adding one feature at a time.

v. Continue adding features until adding more features doesn't improve the model's performance.

```
import pandas as pd
from mlxtend.feature_selection import SequentialFeatureSelector

data = {
    "Gulcose" : [148, 85, 183, 89, 137, 116, 78, 115, 197,
                 125, 110, 168, 139, 189, 166, 100],
    "Pressure" : [72, 44, 66, 64, 40, 74, 50, 0, 70, 96, 72,
                  74, 80, 60, 72, 0],
    "Skin Thickness" : [35, 29, 0, 23, 35, 0, 32, 0, 45,
                        0, 0, 0, 0, 23, 19, 0],
    "BMI" : [33.6, 26.6, 23.3, 28.1, 43.1, 25.6, 31,
             35.3, 30.5, 0, 37.6, 38, 27.1, 30.1,
             25.8, 30],
    "Age" : [50, 31, 32, 21, 33, 30, 26, 29, 53, 54,
             30, 34, 57, 59, 51, 36],
    "Outcome": [1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1,
                1]
}
dataset = pd.DataFrame(data)
dataset.head(3)
```

```
# Seperate x-axis and y-axis data.
x = dataset.iloc[:, :-1]
y = dataset["Outcome"]

# Now we will use classical analysis
from sklearn.linear_model import LogisticRegression

# Initialize the estimator
lr = LogisticRegression()

# Initialize the SequentialFeatureSelector.
fsf = SequentialFeatureSelector(estimator=lr, k_features=5,
                                forward=True)

# fit the selector
fsf.fit(x, y)

# Print the given features name
fsf.feature_names

# Print the selected features
fsf.k_feature_names_

# The top-k accuracy score
fsf.k_score_
```

2. Backward Elimination

i. Start with all features: Begin with a model that includes all possible features.

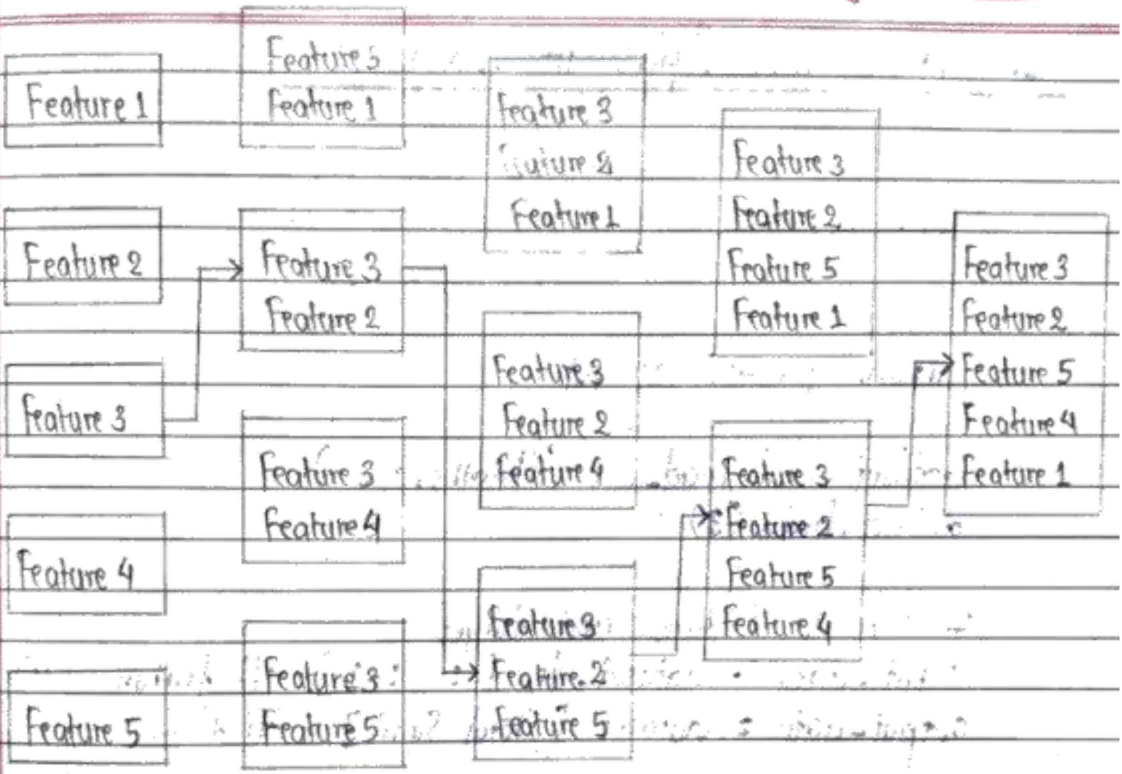
ii. Fit the model : Fit the model using all the features.

iii. Evaluate feature significance : Evaluate the significance of using a choosen metric (e.g. p value in regression)

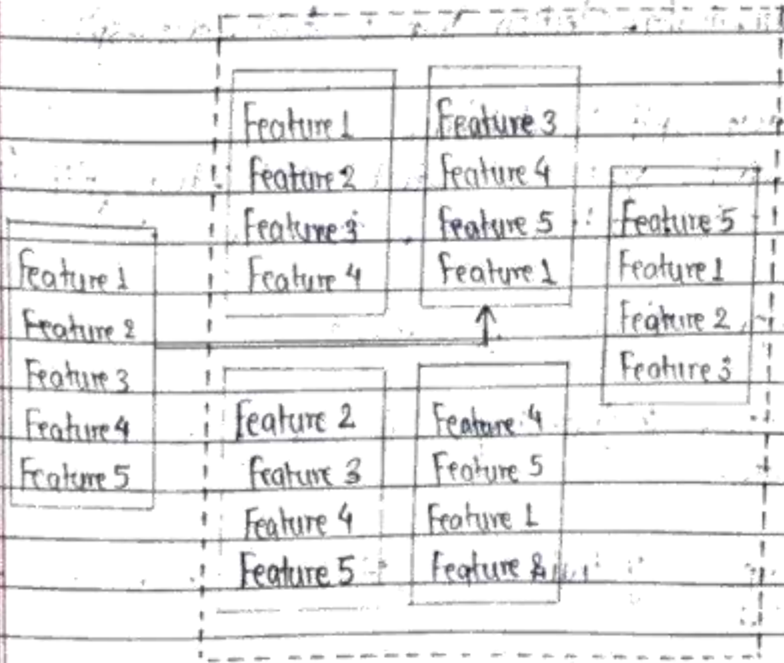
iv. Remove the least significant feature : Identify the feature with the least significance (highest p-value) and remove it from the model.

v. Repeat : Refit the model without the removed feature and repeat the process until all the features are significant..

```
# Initialize the Sequential Feature Selector
fsb = SequentialFeatureSelector(estimator=lr, k_features=5,
                                forward=False)

# fit the selector
fsb.fit(x,y)

# Prints the given feature names
fsb.feature_names

# Prints the selected feature names
fsb.k_feature_names_

# The top-k accuracy score
fsb.k_score_
```



FORWARD ELIMINATION

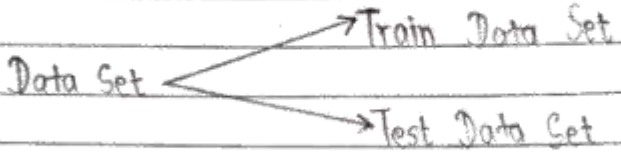


It stops here because there is no improvement found by our model on moving onword. It also happens in forward elimination.

BACKWARD ELIMINATION

4. Supervised Learning in ML

1. Train Test Split in Dataset




```
import pandas as pd

dataset = pd.read_csv("Maths.csv")
dataset.head(3)

# Seperate features and target.
input_data = dataset.iloc[:, :-1] # features (x)
output_data = dataset["Total Sum"] # target (y)

# Import train_test_split from sklearn.model_selection
from sklearn.model_selection import train_test_split

# Train test split in database.
x_train, x_test, y_train, y_test = train_test_split(input_data, output_data, test_size = 0.25)

dataset.shape # size of dataset

x_train.shape, x_test.shape # size of train and test for input_data.

y_train.shape, y_test.shape # size of train and test for output_data.
```

2. Regression Analysis

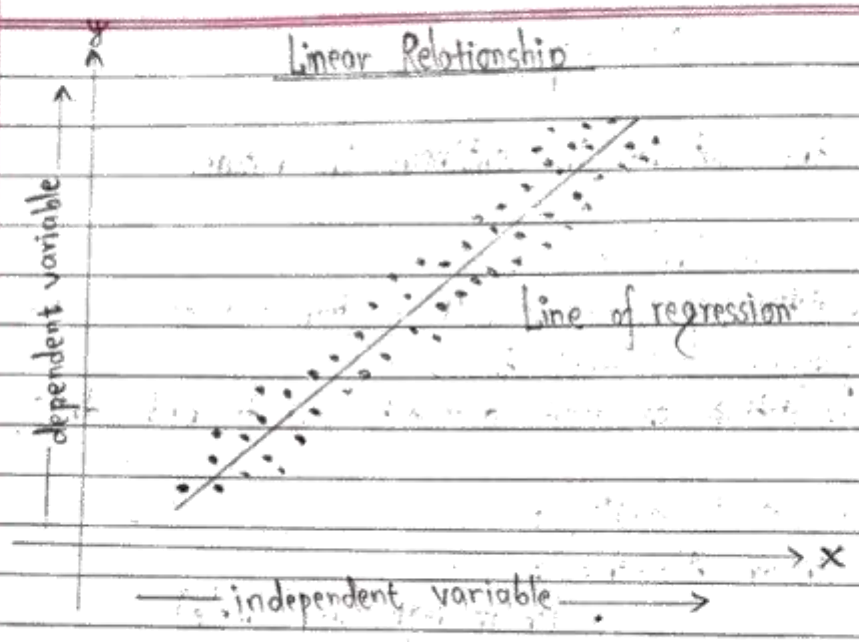
- If the outcome is continous in nature.

- Real World Applications :
  - Prediction of rain using tempreature and other factors.
  - Determining market trends.
  - Prediction of road accidents due to rash driving

- Types of Recognition :
  1. Linear Algorithms :
     - Linear regression (simple)
     - Multi-linear regression
     - Lasso regression
     - Ridge regression
  2. Non-linear Algorithms :
     - Polynomial regression
     - Decision tree regression
     - Random forest regression
     - Support vector regression
     - K-nearest neighbour regression

3. Linear Regression (Simple)

- Simple Linear Regression is a type of regression algorithms that models the relationship between a dependent variable and a single independent variable.
  Example: scholarship is dependent on marks linearly.



Linear Equation:

$$y = mx + c$$

Where,

y = dependent variable
x = independent variable
m = slope / gradient / coefficient
c = intercept

Working Practically in Jupyter:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

data = {
    "Score" : [689, 512, 782, 742, 694, 789, 673, 675,
```

```
        609, 831, 532, 661],
"Prize": [3060, 1980, 3050, 3270, 3370, 2990, 2600,
          2480, 2310, 3510, 1860, 2600],
}
dataset = pd.DataFrame(data)
dataset.head(3)

dataset.isnull.sum() # Check for null values first

x=dataset[["Score"]] # x should be in two dimension
y= dataset["Prize"] # y should be in one dimension.
x.ndim, y.ndim # Shows dimension

# Check if the data follows linear regression by graph
sns.scatterplot(x="Score", y="Prize", data=dataset)
plt.show()

# Train test split
x_train, x_test, y_train, y_test = train_test_split(x, y,
    test_size=0.2, random_state=42)
#random_state ensures that the same random splits are
# generated everytime you run the code.

# Creating a simple linear model
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(x_train, y_train) # Train model on data.

lr.predict([[689]]) # New prediction by model.
```

```
# Check accuracy score
lr.score (x_test, y_test)

# Get coefficient (m) and intercept (c) of our linear
# model in y = mx + c
lr.coef_ , lr.intercept_

# Test the prediction manually if it is true based on
# model.
output = lr.coef_ * 689 + lr.intercept_  #y=mx+c
output  # Output will be same as predicted by model

# Check prediction line in our data by graph
sns.scatterplot (x = "Score" , y = "Prize", data = dataset)
plt.plot (dataset ["Score"], lr.predict (x), color = "red")
plt.legend (["Original Data" , "Prediction Line"])
# To save graph
plt.savefig ("SavedGraph.jpg")
# To show graph
plt.show()
```

4. Multi-Linear Regression

- Multiple linear regression is an extension of simple linear regression as it takes more than one predicator variable to predict the response variable.

Example: Prediction of salary based on age and experience of employee.

Mathematically, in multiple linear regression

$$y = m_1x_1 + m_2x_2 + m_3x_3 + \dots + m_nx_n + c$$

Working practically in Jupyter:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

data = {
    "Age" : [i for i in range (25,50)],
    "Experience" : [i for i in range (25)],
    "Salary" : [35000+i*1000 for i in range (25)],
}

dataset = pd.DataFrame (data)
dataset.head ()

dataset.shape ()

dataset.isnull ().sum () # Check and fill null values first

#Visualization of data
sns.pairplot (data=dataset)
plt.show ()
"The graphs are highly linear"

sns.heatmap (data=dataset.corr (), annot = True)
plt.show ()
"High correlation is found so, Highly linear"
```

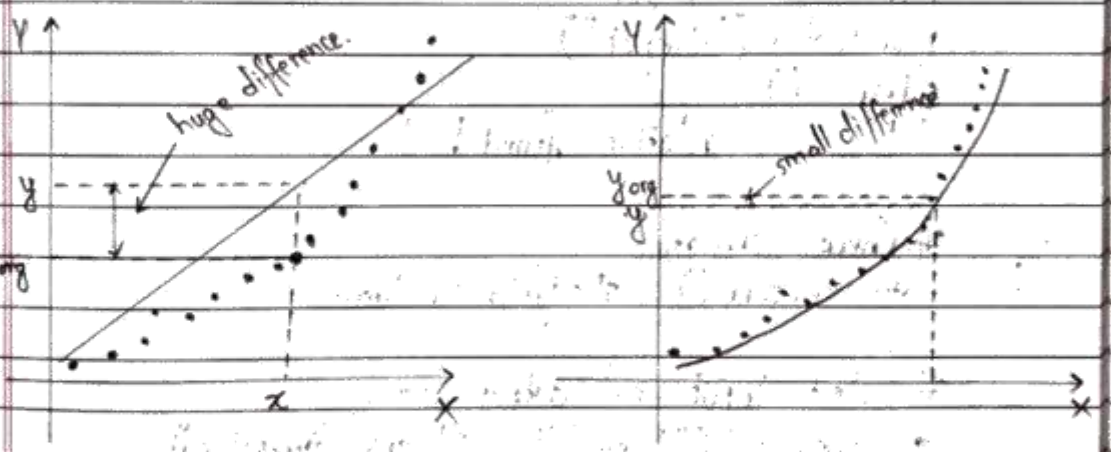
```
# Select input and output
x = dataset.iloc[:, :-1]
y = dataset["Salary"]

# Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y,
    test_size = 0.2, random_state = 42)

# Build the model
from sklearn.linear_model import LinearRegression

x.ndim  # Two dimensional

lr = LinearRegression()
lr.fit(x_train, y_train)  # y = m1x1 + m2x2 + c

# Test the model
lr.score(x_test, y_test)

# New predictions
lr.predict([[50, 25]])  # [[Age, Experience]]

lr.coef_, lr.intercept_  # y = m1x1 + m2x2 + c
# You will get values of m1, m2 and c.
# By using these values you can also do manual
# calculations.
```

5. Polynomial Regression

- Polynomial regression is a regression algorithm that models that models a relationship between a dependent (y) and independent (x) as n^{th} degree polynomial.

- Mathematically,

$$y = b_0 + b_1 x_1 + b_2 x_1^2 + b_3 x_1^3 + \dots b_n x_1^n$$



Simple linear model

Polynomial model

Working practically in Jupyter:

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

data = {
    "Input" : [i for i in range(100)],
    "Output" : [3*i**2 + 5*i**3 for i in range(100)],
}
```

```
dataset = pd.DataFrame(data)
dataset.head(3)

# Check and remove null values
dataset.isnull().sum()

# Visualize data
plt.scatter(dataset["Input"], dataset["Output"])
plt.xlabel("Input")
plt.ylabel("Output")
plt.show()
"Polynomial relation found!"

# Check correlation
datascore.corr()  # Highly correlated

# Select input and output data.
x = dataset[["Input"]]  # Two dimensional
y = dataset["Output"]  # One dimensional

# Convert the data to polynomial nature.
from sklearn.preprocessing import PolynomialFeatures

pf = PolynomialFeatures(degree=3)  # degree is max power of x
# which you want to follow.
pf.fit(x)
x = pf.transform(x)  # Returns an array (2D).

# Train test split
from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split (x, y,
                    test_size = 0.2, random_state = 42)

# Build the polynomial model
from sklearn.linear_model import LinearRegression

lr = LinearRegression ()
lr.fit (x_train, y_train)

# Test the model
lr.score (x_test, y_test)

# New predictions
lr.predict (pf.transform ([[123]])) # New data should be
# in polynomial form

# Visualize prediction line
plt.scatter (dataset ["Input"], dataset ["Output"])
plt.plot (dataset ["Input"], lr.predict (pf.transform (dataset
                    [["Input"]])), color = "red")
plt.xlabel ("Input")
plt.ylabel ("Output")
plt.show ()
```

6. Cost Function

- A cost function is an important parameter that determines how well a machine learning model performs for a given dataset.

↳ Cost function is a measure of how wrong the model is in estimating the relationship between x (input) and y (output) parameter.

Types of Cost Function

Regression cost function
Classification cost function
- ↳ Binary classification cost functions.
- ↳ Multi-class classification cost functions.

1. Regression Cost Function

Regression models are used to make a prediction for the continous variables.

- MSE (Mean Square Error)
- RMSE (Root Mean Square Error)
- MAE (Mean Absolute Error)
- R^2 Accuracy.

2. Binary Classification Cost Functions

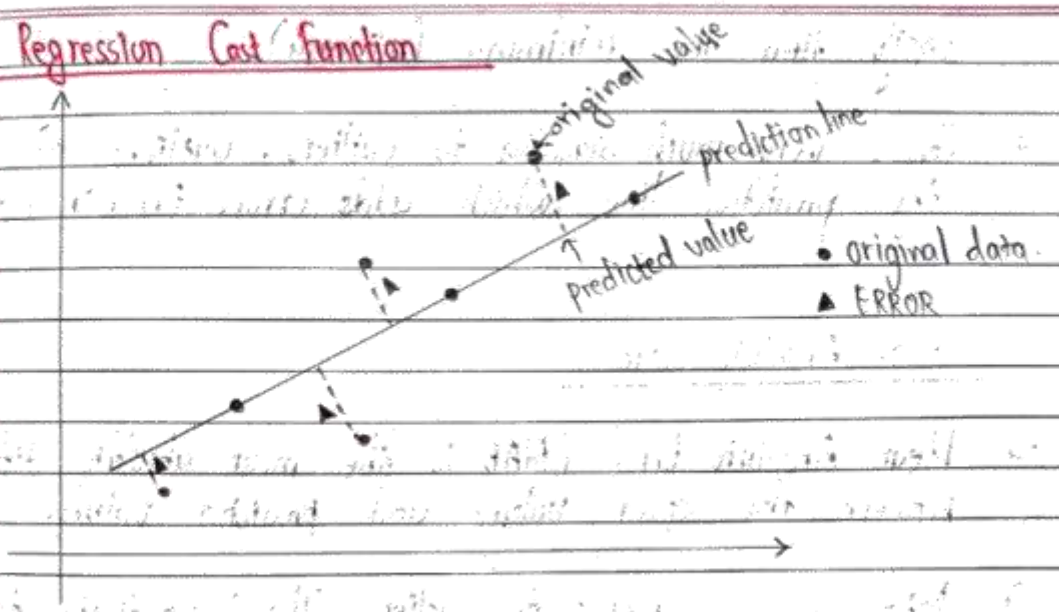
Classification models are used to make predictions of categorical variables such as predictions for 0, or 1, cat or dog, True or false, etc.

3. Multi-class Classification Cost Functions

A multi-class classification cost function is used in the classification problems for which instances are allocated to one of more than two classes.

- Binary Cross Entropy Cost Function or Log Loss Function.

1. Regression Cost Function



Mean Square Error :

- Mean Squared Error (MSE) is the mean squared difference between the actual and predicted values. MSE penalizes high errors caused by outliers by squaring the errors.
- Mean Squared Error is also known as L2 loss.
- Mathematically,

$$\text{Mean Squared Error} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Where,

y = original value
$\hat{y}$ = predicted value.
n = no. of rows.

- Pros : It is a quardatic differential equation so, we can

easily find the minimum (minima).

- Cons: Very much sensitive to outliers. Outliers can change the prediction line which adds error. Error in square.

Mean Absolute Error

- Mean Absolute Error (MAE is the mean absolute difference between the actual values and predicted values.
- MAE is more robust to outliers. The insensitivity to outliers is because it doesn't penalize high errors caused by outliers.
- Mathematically,

$$\text{Mean Absolute Error} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

where,

y = original value
$\hat{y}$ = predicted value
n = no. of rows

- Pros: Error in original value. Insensitive to outliers than Mean Square Error.
- Cons: It is not a differential equation as $|x|$ is not differential.

Root Mean Squared Error

- Root Mean Squared Error (RMSE) is the root squared mean of the difference between actual and predicted values.
- RMSE can be used in situations where we want to penalize high errors but not as much as MSE does.
- Mathematically,

$$\text{Root Mean Squared Error} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Finding Best Fit Line (Gradient Descent Technique)

- Gradient Descent is an optimization algorithm used to minimize the cost function in machine learning models. It iteratively adjusts the model parameters in the direction of the negative gradient of the cost function to find optimal set of parameters.

1. **Initialize Parameters:** Start with initial guesses for the model parameters (weights and biases)
2. **Compute the gradient:** Calculate the gradient of the cost function with respect to each parameter. The gradient is a vector of partial derivatives that points in the direction of the steepest ascent of the cost function.

3. Update Parameters: Adjust the parameters in the opposite direction of the gradient.
The update rule is:

$$\theta = \theta - \alpha \nabla J(\theta)$$

where,

θ represents the parameters.
α is learning rate
$\nabla J(\theta)$ is the gradient of cost function.

The same formula can be written as:

$$m_{new} = m_{old} - \lambda \left(\frac{dL}{dm}\right)$$

Where, λ is learning rate
L is cost function

4. Repeat: Repeat the process until the cost function converges to minimum or a predefined number of iteration is reached.

7. Regularization

- This is a form of regression, that constrains / regularizes or shrinks the coefficient estimates towards zero.
- This technique discourages learning a more complex or flexible model, so as to avoid the risk of overfitting
- Regularization techniques are:
  1. Lasso regularization / L1
  2. Ridge regularization / L2

1. Lasso Regularization (L1)

- This is a regularization technique used in feature selection using a Shrinkage method also referred to as the penalized regression method.
- Lasso Regression magnitude of coefficients can be exactly zero.
- Mathematically,

$$\text{cost function} = \text{loss} + \lambda \Sigma |w|$$

where,

loss = sum of squared residual
λ = penalty
w = slope of the curve.

2. Ridge Regularization (L2)

- Ridge regression, also knows as L2 regularization, is an extension to linear regression that introduces a regularization term to reduce model complexity and help prevent overfitting.
- Ridge Regression is working value / magnitude of coefficients is almost equal to zero.
- Mathematically,

$$\text{cost function} = \text{loss} + \lambda \Sigma |w|^2$$

where,

loss = sum of residual

λ = penalty
w = slope of the curve.

Working Regularization Practically in Jupyter :

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandScalar
from sklearn.model_selection import train_test_split

dataset = pd.read_csv("Maths.csv")
dataset.head(3)

# Check for null values
dataset.isnull().sum()

# Fill all the null values
for column in dataset.columns:
    dataset[column].fillna(dataset[column].mean(),
                           inplace=True)

# for visualization
sns.heatmap(data=dataset.corr(), annot=True)
plt.show()  # heatmap of correlation among columns.

# Select dependent and independent variables
x = dataset.iloc[:, :-1]
y = dataset["Total Sum"]
```

Date Page 105

```
# Standardize features by removing the mean and scaling
# to unit variance.
sc = StandardScalar()
sc.fit(x)
x = pd.DataFrame(sc.transform(x), columns = x.columns)
x.head(5)

# Train test split
x_train, x_test, y_train, y_test = train_test_split(x, y,
                    test_size = 0.2, random_state = 42)

from sklearn.linear_model import LinearRegression, Lasso,
                                 Ridge
from sklearn.metrics import mean_absolute_error,
                            mean_squared_error
import numpy as np
```

Linear Regression :

```
lr = LinearRegression()
lr.fit(x_train, y_train)
lr.score(x_test, y_test)

# View regression cost
print(mean_squared_error(y_test, lr.predict(x_test)))
print(mean_absolute_error(y_test, lr.predict(x_test)))
print(np.sqrt(mean_squared_error(y_test, lr.predict(
                x_test)))) # Root mean squared error.

# Visualize linear regression
plt.figure(figsize = (5,5))
```

```
plt.bar (x.columns , lr.coef_)
plt.title ("Linear Regression")
plt.xlabel (" columns")
plt.ylabel (" coef_")
plt.show ()
```

Lasso :

```
la = Lasso (alpha = 0.01)
la.fit (x_train , y_train)
la.score (x_test, y_test)

# View regression cost
print ( mean_squared_error (y_test , la. predict (x_test)))
print ( mean_absolute_error (y_test , la. predict (x_test)))
print (np.sqrt ( mean_squared_error (y_test , la. predict (
        x_test )))) # Root mean squared error

# Visualize Lasso
plt. figure (figsize = (5, 5))
plt. bar (x.columns , la.coef_)
plt.title ("Lasso")
plt.xlabel (" columns ")
plt.ylabel (" coef_")
plt.show()
```

Ridge :

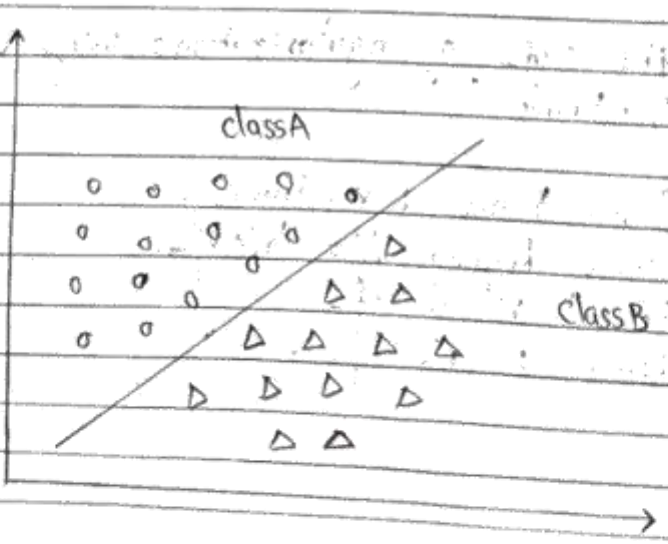
```
ri = Ridge (alpha = 0.01)
ri.fit (x_train , y_train )
ri. score (x_test , y_test )
```

```
# View regression cost
print( mean_squared_error (y_test , la.predict (x_test)))
print (mean_absolute_error ( y_test , la. predict (x_test)))
print (np.sqrt (mean_squared_error (y_test , la.predict (
                x_test)))) # Root mean squared error.

# Visualize ridge
plt.figure (figsize = (5,5))
plt.bar (x.columns, ri_coef_)
plt.title ("Ridge")
plt.xlabel ("columns")
plt.ylabel ("coef_")
plt.show()

# Compare the coef_ of regularizations using Data frame
df = pd. Dataframe (
    {
        "Column Name" : x.columns,
        "Linear Regression" : lr.coef_,
        "Lasso" : la.coef_,
        "Ridge" : ri.coef_,
    }
)
```

5. Classification in ML

1. Classification Algorithm

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



Types of Classifications

1. Binary Classifier: If the classification problem has only two possible outcomes, then it is called as Binary Classifier.
   Example: SPAM or NOT SPAM, CAT or DOG, etc.

2. Multi-class Classifier: If a classification problem has more than two outcomes, then it is called as Multi-class Classifier.
Example: Classification of types of crops, Classification of types of music, etc.

Types of ML Classification Algorithms

Non-linear Models
- K-Nearest Neighbours
- SVM (Support Vector Machines)
- Naive Bayes
- Decision Tree Classification
- Random forest Classification

Linear Models
- Logistic Regression
- Support Vector Machines

2. Evaluating a Classification Model

1. Log Loss or Cross-Entropy Loss
2. Confusion Matrix
3. AUC - ROC curve

Real World Examples of Classification Problems:

- Disguishing the spam mails and non-spam mails.
- Classification of documents.
- Distinguish between animals.

3. Logistic Regression

- Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique.
- It is used for predicting the categorical dependent variable using a given set of independent variables.
- Therefore, the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, True or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.

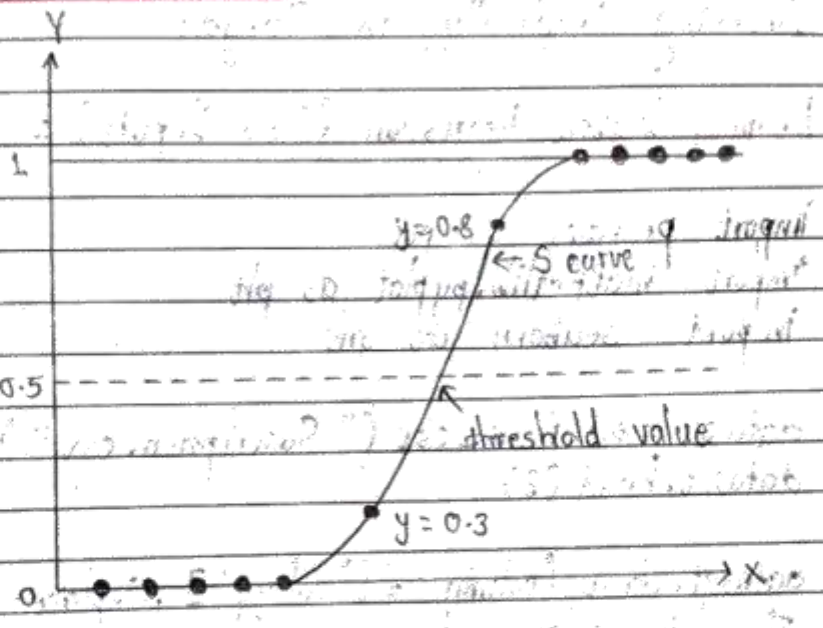
Types of Logistic Regression

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

1. **Binomial**: In binomial logistic regression, there can be only two possible types of dependent variables, such as 0 or 1, Pass or fail, etc.
2. **Multinomial**: In multinomial logistic regression, there can three or more possible unordered types of the dependent variable, such as "cat", "dog", or "sheep"
3. **Ordinal**: In ordinal logistic regression, there can be 3 or more possible ordered types of dependent

variables, such as "low", "Medium", "High". e.

Sigmoid Function



The logistic regression equation can be obtained from the linear Regression equation.
Mathematically, Logistic regression equation is:

$$y = \frac{1}{1+e^{-x}}$$

where,

y = dependent variable
x = independent variable
e = euler's constant

Uses of Logistic Regression

i. Spam email filtering

ii. Object classification
iii. Credit card fraud detection.

Working Practically in Jupyter

Binomial Logistic Regression (Two Inputs) :

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

dataset = pd.read_csv("Suscription.csv")
dataset.head(3)

dataset.drop(columns=["Salary"], inplace=True)
# Delete salary column
dataset.head(3)

# Visualize the data
plt.figure(figsize=(4,3))
sns.scatterplot(x="Age", y="Purchase", data=dataset)
plt.show()

# Select dependent and independent variables.
x = dataset[["Age"]]
y = dataset["Purchase"]

# Train test split
from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(x,
        y, test_size=0.2, random_state=42)

# Implement model
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
lr.fit(x_train, y_train)  # Train model

lr.score(x_test, y_test)  # Test model

lr.predict([[40]])  # New prediction

# Visualize prediction line.
plt.figure(figsize=(9,3))
sns.scatterplot(x="Age", y="Purchase", data=dataset)
sns.lineplot(x="Age", y=lr.predict(x), data=dataset,
             color="red")
plt.show()
```

Binomial Logistic Regression (Multiple Inputs):

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

dataset = pd.read_csv("Suscription.csv")
dataset.head(3)

# Graphical visualization
plt.figure(figsize=(5,4))
```

```
sns.scatterplot(x="Age", y="Salary", data=dataset,
                hue="Purchase")
plt.legend(loc=2)
plt.show()

# Dependent and Independent data.
x = dataset.iloc[:, :-1]
y = dataset["Purchase"]

# Train test split.
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(
        x, y, test_size=0.2, random_state=42)

# Train your model.
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
lr.fit(x_train, y_train)

lr.score(x_test, y_test)

lr.predict([[36, 35000]])

# Visualization of prediction line
from mlxtend.plotting import plot_decision_regions

plot_decision_regions(x.to_numpy(), y.to_numpy(),
                      clf=lr)
plt.show()
```

```
# Coefficients of logistic regression equation
lr.coef_

# Constant of logistic regression equation
lr.intercept_
```

Binomial Logistic Regression (Using PolynomialFeatures):

- This trains model on polynomial equation instead of linear.
- This is used for better accuracy of model.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

dataset = pd.read_csv("Suscription.csv")
dataset.head(3)

sns.scatterplot(x="Age", y="Salary", data=dataset,
                hue="Purchase")
plt.show()

# Seperate dependent and independent datas.
x = dataset.iloc[:, :-1]
y = dataset["Purchase"]

# Polynomialize the independent data
from sklearn.preprocessing import PolynomialFeatures
```

```
pf = PolynomialFeatures (degree = 2)
pf. fit (x)
x = pd. DataFrame (pf. transform (x))
x. head (3)

# Train test split
from sklearn. model_selection import train_test_split

x_train , x_test , y_train , y_test = train_test_split (x,
        y, test_size = 0.2 , random_state = 42 )

# Build the polynomial model.
from sklearn. linear_model import LogisticRegression

lr = LogisticRegression ()
lr. fit (x_train , y_train)

lr. score (x_test , y_test)
```

- Multinomial Logistic Regression :

```
import pandas as pd
import matplotlib. pyplot as plt
import seaborn as sns

dataset = pd. read_csv ("Irisflower. csv")
dataset. head (3)

dataset ["Species"]. unique() # Check for unique values
```

```
# Visualizing dataset
sns.pairplot(data=dataset, hue="Species")
plt.show()

# Seperate dependent and independent data
x = dataset.iloc[:, :-1]
y = dataset["Species"]

# Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x,
        y, test_size=0.2, random_state=42)

# Build model
from sklearn.linear_model import LogisticRegression

""" OVR Method """

lr = LogisticRegression(multi_class="ovr")
lr.fit(x_train, y_train)

lr.score(x_test, y_test)

""" Using Multinomial """

lr = LogisticRegression(multi_class="multinomial")
lr.fit(x_train, y_train)

lr.score(x_test, y_test)
```

4. Confusion Matrix

- A confusion matrix is a simple and useful tool for understanding the performance of a classification model, like one used in machine learning or statistics.
- It helps you evaluate how well your model is doing in categorizing things correctly.
- It is also known as the error matrix.
- The matrix consists of prediction result in a summarized form, which has a total number of correct predictions and incorrect predictions.

Sr. No.	Actual	Predicted
1	1	1
2	0	0
3	1	0
4	0	1
5	0	0
6	0	1
7	1	1

Now, the confusion matrix for the data in the table can be created as follows:

Date Page 11.3

Confusion Matrix

Actual Labels	0	1
0	TN = 86	FP = 2
1	FN = 1	TP = 2

Predicted Labels

TN → true negative
FP → false positive
FN → false negative
TP → true positive

- Mathematically,
  - Accuracy = $\frac{TN + TP}{TN + TP + FP + FN}$

  i.e. Accuracy = $\frac{TN + TP}{N}$, N = total number of data.

  - Error = $\frac{FN + FP}{N}$

- **False Negative:** The model has predicted No, but the actual value was Yes.
  It is also called as Type-II error.

- **False Positive:** The model has predicted Yes, but the actual value was No.
  It is also called Type-I error.

Precision

- It helps us to measure the ability to classify positive samples in model.
- Mathematically,

$$\text{Precision} = \frac{TP}{TP + FP}$$

- Example:

A

	Detected Cancer	Not Detected Cancer
Has Cancer	1000	200
No cancer	800	8000

B

	Detected Cancer	Not Detected Cancer
Has Cancer	1000	500
No Cancer	500	8000

Here, we find that has cancer but not detected is a major issue which is type I error (FP error). To minimize FP error we try to keep the value of precision high by minimizing value of FP.

In this example model B will be selected though having same accuracy as value of FP is less in model B.

Remember that to minimize Type I error, we should always try to increase value of precision. The max value of precision is 1 when FP = 0.

Date Page 121

Recall :

- It helps us to measure how many positive samples were correctly classified by the ML model.
- Mathematically,

$$\text{Recall} = \frac{TP}{TP + FN}$$

- Example:

A

	Sent to spam	Not sent to spam
Spam	100	170
Not spam	30	700

B

	Sent to Spam	Not sent to spam
Spam	100	190
Not spam	10	700

Here, we find that not spam mails sent to spam is a major issue which is type II error (FN error). To minimize FN error we try to keep the value of recall high by minimizing value of FN.

In this example model B will be selected though having same accuracy as value of FN is less in model B.

Remember that to minimize type II error, we should always try to increase value of recall. The max value of recall is 1 when FN = 0.

F1 - Score

- It is harmonic mean of precision and recall. It takes both false positive and false negative into account. Therefore, it performs well on an imbalanced dataset.

- Mathematically,

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Working Practically

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

dataset = pd.read_csv("Suscription.csv")
dataset.head(3)

# Seperate dependent and independent variables
x = dataset.iloc[:, :-1]
y = dataset["Purchase"]

# Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x,
    y, test_size = 0.2, random_state = 42)
```

Date
Page 121

```
# Build your model
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
lr.fit(x_train, y_train)

lr.score(x_test, y_test)

# Create a confusion matrix
from sklearn.metrics import confusion_matrix, precision_score,
                            recall_score, f1_score

# You need training data and predicted data to make a
# confusion matrix
cf_mat = confusion_matrix(y_test, lr.predict(x_test))
cf_mat

# For graphical visualization
sns.heatmap(cf_mat, annot=True)
plt.show()

# Precision
precision_score(y_test, lr.predict(x_test))

# Recall
recall_score(y_test, lr.predict(x_test))

# F1 score
f1_score(y_test, lr.predict(x_test))
```

5. Imbalanced Dataset Handling

1. Random Under Sampling

- We will reduce the majority of the class so that it will have same no of as the minority.
- Random samples from majority class is taken.

2. Random Over Sampling

- We will increase the size of minority i.e. inactive class to size of majority class i.e. active class.
- Copies of minority class is formed.

Working Practically

```
import pandas as pd

dataset = pd.read_csv("Suscription.csv")
dataset.head(3)

# Check if the data is imbalanced.
dataset["Purchase"].value_counts()
# The dataset is unbalanced.

""" Imbalanced dataset leads to biasing of the machine learning module. So, we need to balance the dataset for avoiding this problem. """
```

```
# Seperate dependent and independent variables
x = dataset.iloc[:, :-1]
y = dataset["Purchase"]
```

Random Under Sampling:

```
# Balance the dataset using random under sampling.
from imblearn.under_sampling import RandomUnderSampler

rus = RandomUnderSampler()
rus_x, rus_y = rus.fit_resample(x, y)

rus_y.value_counts() # Data is now balanced.

# Train Test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(rus_x,
    rus_y, test_size = 0.2, random_state = 42)

# Build the model
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
lr.fit(x_train, y_train)

lr.score(x_test, y_test)
```

Random Over Sampling:

```
# Balance the dataset using random over sampling.
```

```
from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler()
ros_x, ros_y = ros.fit_resample(x, y)

ros.y.value_counts()  # Data is now balanced

# Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(ros_x,
        ros_y, test_size = 0.2, random_state = 42)

# Build the model
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
lr.fit(x_train, y_train)

lr.score(x_test, y_test)
```

6. Naive Bayes Algorithm

- Naive Bayes is a classification algorithm based on Baye's theorem.

- Bayes' Theorem : is a probability theory that describes the probability of an event, based on prior knowledge of conditions that might be related to the event.

- Naive : It is called Naive because it assumes that the occurrence of a certain feature is independent of the occurrence of other features.
- 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.

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

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

Types of Naive Bayes Model

There are three types of Navie Bayes Model:

1. Gaussian
2. Multinomial
3. Bernoulli

1. Gaussian Naive Bayes:

- Assume that continous features follow a Gaussian (normal) distribution.
- Suitable for features that are continous and have a normal distribution.

2. Multinomial Naive Bayes:

- Assumes that features follow a multinomial distribution.
- Typically used for discrete data, such as text data, where each feature represents the frequency of a term.

3. Bernoulli Naive Bayes:

- Assumes that features are binary (boolean) variables.
- Suitable for data that can be represented as binary features, such as document classification problems where each term is either present or absent.

Working Practically:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from mlxtend.plotting import plot_decision_regions

dataset = pd.read_csv("Suscription.csv")
dataset.head(3)

# Check for null values
dataset.isnull().sum()  # No null values

# Visualize data.
plt.figure(figsize = (5,5))
sns.scatterplot(x="Age", y="Salary", data=dataset,
                hue="Purchase")
plt.show()

# Seperate dependent and independent variables
x = dataset.iloc[:, :-1]
y = dataset["Purchase"]

# Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y,
        test_size=0.2, random_state=42)
```

Gaussian Naive Bayes:

```
# Build model
from sklearn. naive-bayes import GaussianNB

gnb = GaussianNB()
gnb. fit (x-train , y-train)

# Visualize the decision region
plt. figure (figsize = (5,5))
plot-decision_regions (x. to-numpy(), y. to-numpy(),
                       clf. gnb )
plt.show ()

gnb. score (x-test , y-test)
```

Multinomial Naive Bayes :

```
# Build model
from sklearn. naive_bayes import MultinomialNB

mnb = MultinomialNB()
mnb. fit (x-train, y-train)

mnb. score (x-test , y-test )

# Visualize the decision region
plt. figure (figsize = (5,5))
plot- decision_regions (x. to-numpy(), y. to-numpy(),
                        clf = mnb )
plt. show ()
```

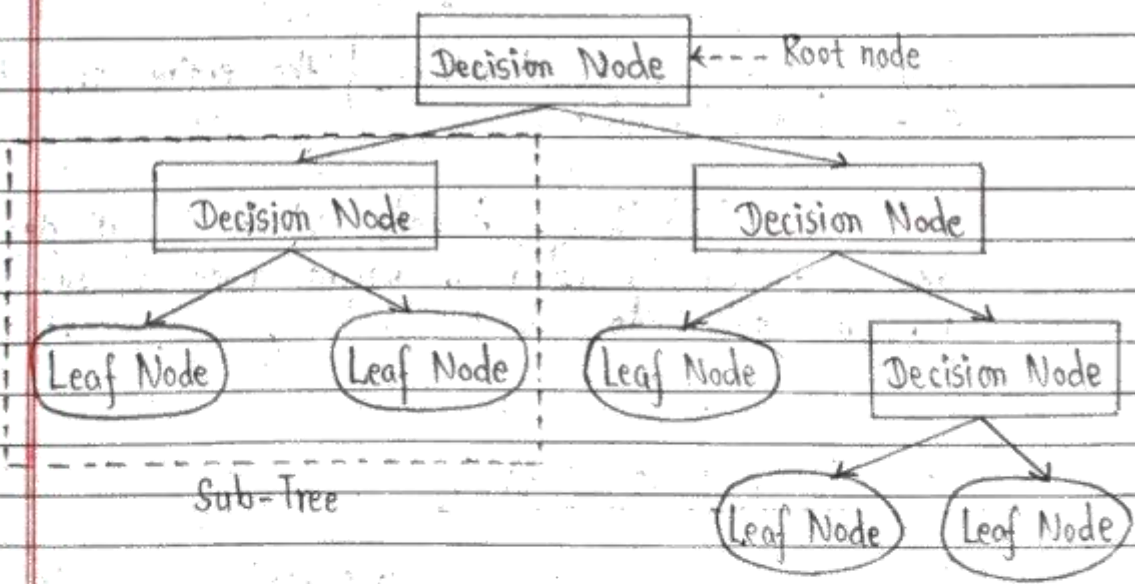
Bernoulli Naive Bayes :

```
# Build model
from sklearn.naive_bayes import BernoulliNB

bnb = BernoulliNB()
bnb.fit(x_train, y_train)

bnb.score(x_test, y_test)

# Visualize the decision region
plt.figure(figsize=(5,5))
plot_decision_regions(x.to_numpy(), y.to_numpy(),
                      clf=bnb)
plt.show()
```

6. Non-Linear Supervised Algorithm in ML

1. Decision Tree (Classification)

- Decision Tree is a supervised learning technique that can be used for both classification and regression problems, but mostly it is preferred for solving classification problems.
- In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm.



Terminologies

Root Node : It represents the entire population or sample, and this further gets divided into two or more homogenous sets.

Splitting: It is a process of dividing a node into two or more sub-nodes.

Decision Node: When a sub-node splits into further sub-nodes, then it is called the decision node.

Leaf / Terminal Node: Nodes do not split is called Leaf or Terminal node.

Pruning: When we remove sub-nodes of a decision node, this process is called pruning. It can be said the opposite process of splitting.

Branch / Sub-Tree: A subsection of the entire tree is called branch or sub tree.

Parent and Child Node: A node which is divided into sub-nodes is called a parent node of sub-nodes whereas sub-nodes are the child of a parent node.

Attribute Selection Measures

From attribute selection measures, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

1. Information gain
2. Entropy / Gini Index

1. Information Gain:

Information gain is a measurement of changes in

entropy after the segmentation of a dataset based on an attribute. It calculates how much information a feature provide us about a class.
Mathematically,

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

2. Entropy :
Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data.
Mathematically,

$$\text{Entropy}(S) = -P(\text{yes}) \log_2 P(\text{yes}) - P(\text{no}) \log_2 P(\text{no})$$

where,
S = total number of samples
P(yes) = probability of yes
P(no) = probability of no

Gini Index :
Gini Index is measure of impurity or purity used while creating a decision tree in the CART (Classification and Regression Tree) algorithm.
An attribute with low Gini index should be preferred as compared to high Gini index.
Mathematically,

$$\text{Gini Index} = 1 - \sum_{i=1}^{n} P_i^2$$

where,
n = total number of classes in dataset
P_i = probability of an object being classified into class i.

Working Practically

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

dataset = pd.read_csv("Suscription.csv")
dataset.head(3)

# Check for null values.
dataset.isnull().sum()  # No null values.

# Seperate dependent and independent variables.
x = dataset.iloc[:,:-1]
y = dataset["Purchase"]

# Scaling our data.
from sklearn.preprocessing import StandardScaler

stdsc = StandardScalar()
stdsc.fit(x)
x = pd.DataFrame(stdsc.transform(x), columns = x.columns)
x.head(3)

# Train test split.
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y,
                    test_size = 0.2, random_state=42)

# Build decision tree model.
```

```
from sklearn.tree import DecisionTreeClassifier # Classification
#model.

dtc = DecisionTreeClassifier()
dtc.fit(x_train, y_train)

dtc.score(x_test, y_test)

dtc.predict([[19, 19000]])

# Visualization of decision region.
from mlxtend.plotting import plot_decision_regions

plot_decision_regions(x.to_numpy(), y.to_numpy(),
                      clf=dtc)
plt.show()

#Visualization of decision tree
plt.figure(figsize=(50, 50))
plot_tree(dtc)
plt.savefig("DecisionTree.jpg")
plt.show()
```

Handling Overfitting

"Over fitting can be handled either by pre-pruning or post-pruning"

```
# Check if the model is overfitted
dtc.score(x_train, y_train), dtc.score(x_test, y_test)
# Huge difference in score between training and testing data
# means it is overfitted.
```

Pre-Pruning

"In pre-pruning the max_depth is set while creating an object of DecisionTreeClassifier class."

```
dtc = DecisionTreeClassifier(max_depth = 3)
dtc.fit(x_train, y_train)

dtc.score(x_train, y_train), dtc.score(x_test, y_test)
# Less difference is seen now.

# Visualization of decision region.
plot_decision_regions(x.to_numpy(), y.to_numpy(),
                      clf = dtc)
plt.show()

# Visualization of decision tree
plt.figure(figsize = (50, 50))
plot_tree(dtc)
plt.show()
```

Post-Pruning

"It first finds the best max_depth by training the model to different depths and then train the model to selected max depth."

```
# Find the best max_depth
for i in range(1, 20):
    dtc = DecisionTreeClassifier(max_depth = i)
    dtc.fit(x_train, y_train)
    print(dtc.score(x_train, y_train), dtc.score(x_test,
          y_test), i)
```

```
# Now from above output decide the value of max_
# depth.
dtc = DecisionTreeClassifier(max_depth = 4)
dtc.fit(x_train, y_train)

dtc.score(x_train, y_train), dtc.score(x_test, y_test)
# Less difference is seen now.

# Visualization of decision region.
plot_decision_regions(x.to_numpy(), y.to_numpy(),
                      clf=dtc)
plt.show()

# Visualization of decision tree.
plot.figure(figsize=(50, 50))
plot_tree(dtc)
plt.show()
```

2. Decision Tree (Regression)

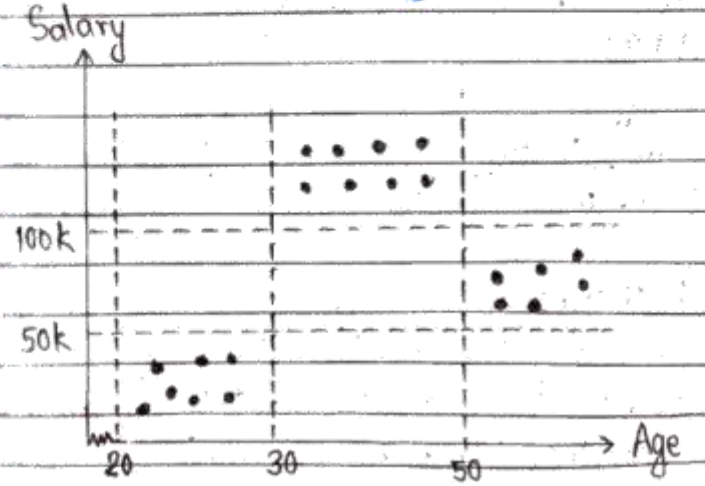


Fig: Graph

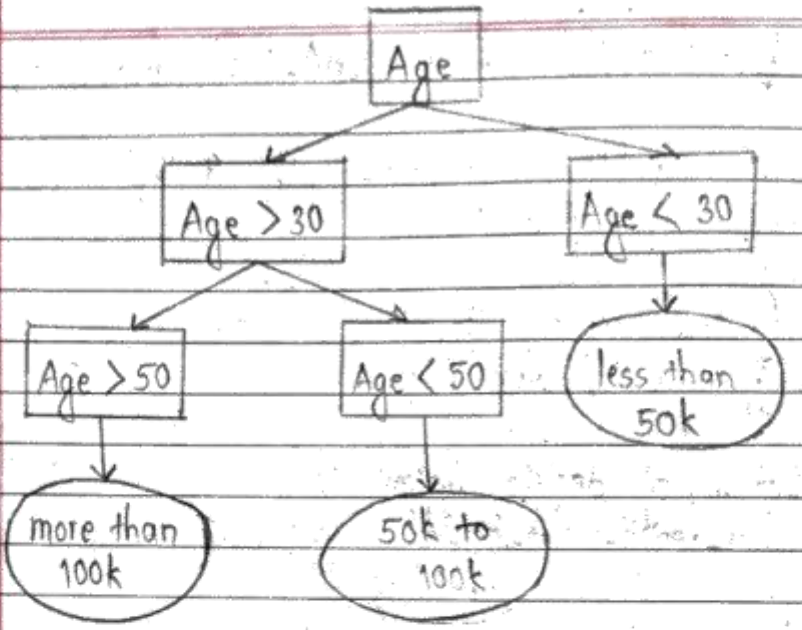


Fig: Decision Tree

Working Practically

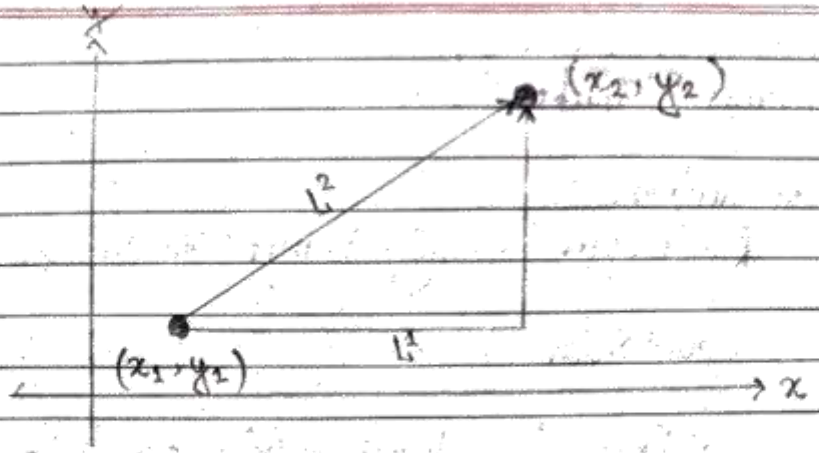
```
import pandas as pd
import matplotlib.pyplot as plt
Import seaborn as sns

dataset = pd.read_csv ("Maths.csv")
dataset.head (3)

# Check for null values
dataset.isnull().sum

# Fill the null values
for column in dataset.columns:
    dataset [column].fillna (dataset [column].mean (),
                             inplace = True )
```

```
# Visualize data.
sns.pairplot(data = dataset)
plt.show()

# Seperate dependent and independent variables.
x = dataset.iloc[:, :-1]
y = dataset["Total Sum"]

# Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y,
        test_size = 0.2, random_state = 42)

# Build a model
from sklearn.tree import DecisionTreeRegressor

dtr = DecisionTreeRegressor()
dtr.fit(x_train, y_train)

dtr.score(x_test, y_test)

# Check if the model is overfitted.
dtr.score(x_train, y_train), dtr.score(x_test, y_test)

# We find huge difference between score of train and
# test data so the model is overfitted.
# Find the best value of max_depth.
for i in range(1, 20):
    dtr = DecisionTreeRegressor(max_depth = i)
    dtr.fit(x_train, y_train)
```

```
print (dtr.score (x_train , y_train), dtr.score (
        x_test , y_test ), i)

# Select the best value of max_depth and train the
# model.
dtr = DecisionTreeRegressor (max_depth = 4)
dtr. fit (x_train , y_train)

# For visualization of decision tree
from sklearn.tree import plot_tree

plt.figure (figsize =(50, 50))
plot_tree (dtr)
plt.show ()
```

3. K-Negrest Neighbours (Classification)

- K-NN algorithm can be used for regression as well as for classification but mostly it is used for classification problems.
- K-NN algorithm is a non-parametric algorithm, which means it doesn't make any assumption on underlying data.
- It is also called a lazy learner algorithm.



Manhattan distance, $L^1 = |x_2 - x_1| + |y_2 - y_1|$
Euclidean distance, $L^2 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

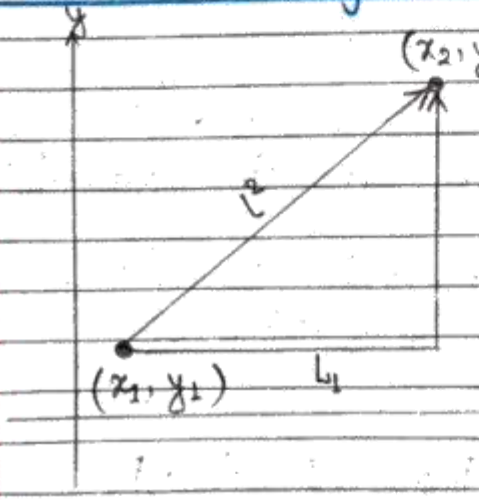
Working practically

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

dataset = pd.read_csv("Suscription.csv")
dataset.head(3)

# Check for null values
dataset.isnull().sum()  # No null values

# Visualize the data
sns.scatterplot(x="Age", y="Salary", data=dataset,
                hue="Purchase")
plt.show()

# Seperate dependent and Independent variables.
```

```
x = dataset.iloc[:, :-1]
y = dataset["Purchase"]

# Data scaling
from sklearn.preprocessing import StandardScaler

ssc = StandardScaler()
ssc.fit(x)
x = pd.DataFrame(ssc.transform(x), columns = x.columns)

# Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x,
    y, test_size = 0.2, random_state = 42)

# Train model from dataset
from sklearn.neighbours import KNeighboursClassifier

knn = KNeighboursClassifier()
knn.fit(x_train, y_train)

knn.score(x_test, y_test)

# Check if the model is overfitted
knn.score(x_train, y_train), knn.score(x_test, y_test)

# Find the best value of n_neighbours
for i in range(1, 30):
    knn = KNeighboursClassifier(n_neighbours = i)
    knn.fit(x_train, y_train)
```

```
print (i, knn.score (x_train, y_train), knn.score
                    (x_test, y_test))

# Train the model by using best value of n_neighbours.
knn = KNeighboursClassifier (n_neighbours = 4)
knn.fit (x_train, y_train)

# Make the new predictions
knn.predict (ssc.fit_transform ([[32, 32000]])) # Use
# scaled data.

# Visualize the decision region
from mlxtend.plotting import plot_decision_regions

plot_decision_regions (x.to_numpy(), y.to_numpy(),
                       clf = knn)
plt.show()
```

4. K-Nearest Neighbours (Regression)



$$L_1^1 = |x_2 - x_1| + |y_2 - y_1|$$

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

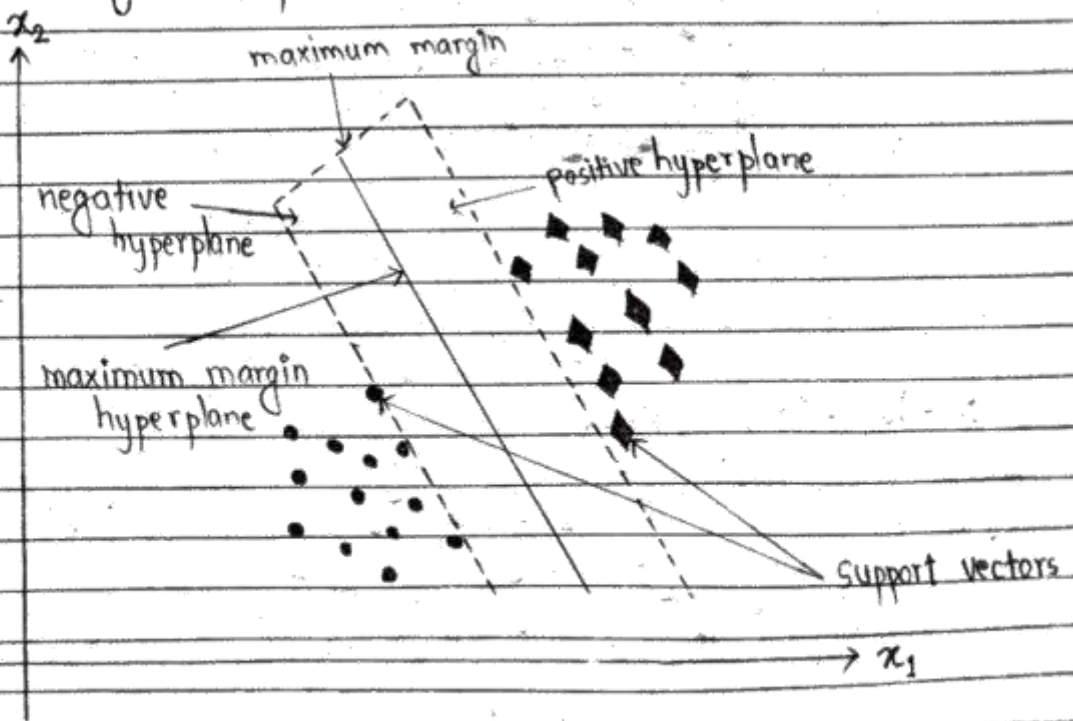
Working practically

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

dataset = pd.read_csv("Maths.csv")
dataset.head(3)

# Check for null values
dataset.isnull().sum()

# Drop all the null values
dataset.dropna(inplace=True)

# Visualize data
sns.pairplot(data=dataset)
plt.show()

# Seperate dependent and independent data
x = dataset.iloc[:, :-1]
y = dataset["Total Sum"]

# Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y,
        test_size=0.2, random_state=42)

# Train your model
from sklearn.neighbours import KNeighboursRegressor
```

```
knn = KNeighbourRegressor()
knn.fit(x_train, y_train)

knn.score(x_test, y_test)

# Check if the model is overfitted.
knn.score(x_train, y_train); knn.score(x_test, y_test)
```

5. Support Vector Machines (Classification)

↳ SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems.

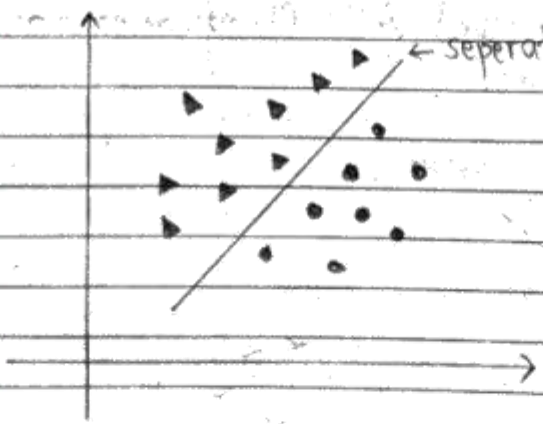


- **Hard Margin :** The algorithm aims to find a hyperplane that perfectly seperates the data into two classes without any misclassifications.

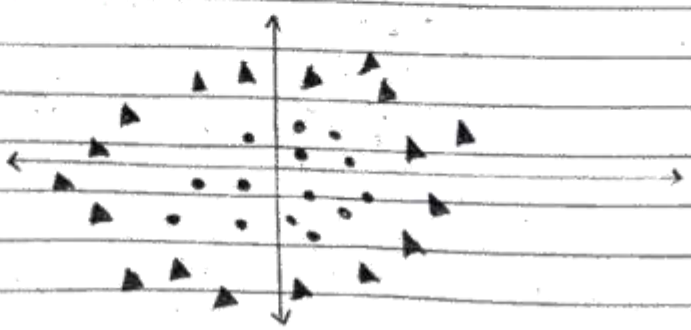
- **Soft Margin:** The algorithm allows for some misclassifications to find a hyperplane that generalizes better to unseen data and is more robust to outliers.

Types of SVM.

1. **Linear SVM:**

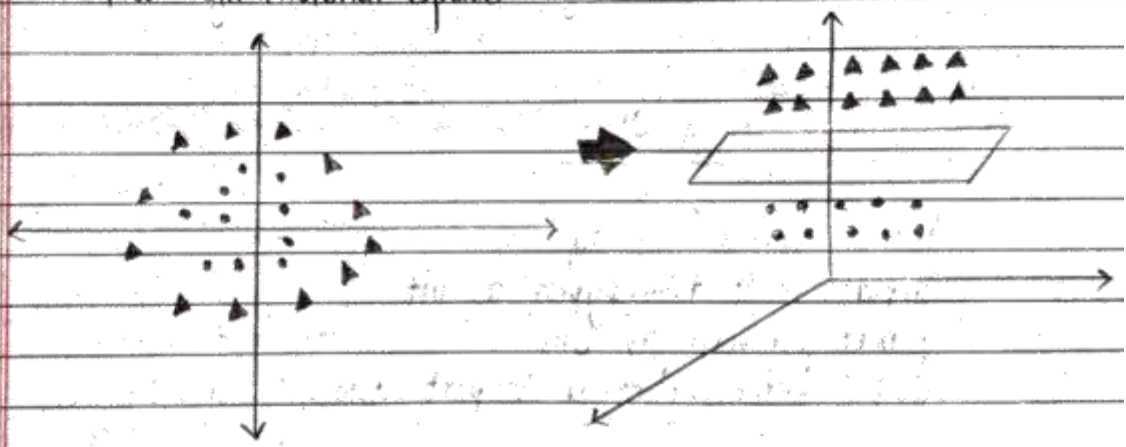


2. **Non-linear SVM.:**



3. Simple SVM:
Typically used for linear regression and classification problems.

4. Kernel SVM:
Has more flexibility for non-linear data because you can add more features to fit a hyperplane instead of a two dimensional space.



Kernel Functions

- Kernal functions play a crucial role in transforming input data into a higher-dimensional space.
- The primary purpose of kernel functions is to allow SVMs to handle non-linearly seperable data by implicitly mapping the input data into a higher dimensional feature space where linear seperation may be more feasible.
- This transformation is done without explicitly calculating

the coordinates of the data points in that higher dimensional space.

↳ Some kernel functions are:-
Linear : $K(w, b) = w^T x + b$
Polynomial : $K(w, x) = (\gamma w^T x + b)^N$
Gaussian RBF : $K(x, x') = \exp(-|x - x'|^2) / 2\sigma^2$
Sigmoid : $K(x, x') = \tanh(\alpha x \cdot x' + \beta)$

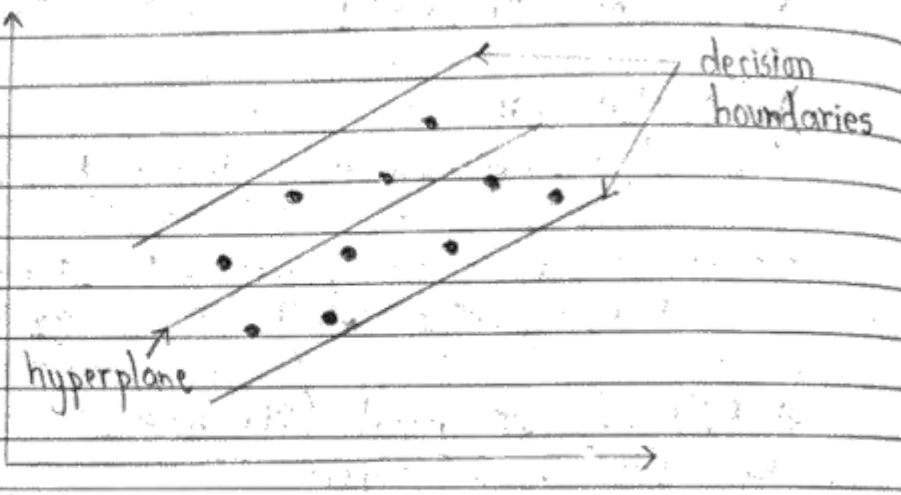
Working Practically:-

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from mlxtend.plotting import plot_decision_regions

dataset = pd.read_csv("Suscription.csv")
dataset.head(3)

# Check for null values
dataset.isnull().sum()  # No null values

#Visualize the data
plt.figure(figsize=(4,3))
sns.scatterplot(x="Age", y="Salary", data=dataset,
                hue="Purchased")
plt.show()

#Seperate Input and output
x = dataset.iloc[:, :-1]
```

```
y = dataset["Purchase"]

# Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y,
        test_size = 0.2, random_state = 8)

# Build the SVM model (Classification)
from sklearn.svm import SVC

svc = SVC(kernel = "linear")
svc.fit(x_train, y_train)

svc.score(x_test, y_test)

# Check if the data is overfitted.
svc.score(x_train, y_train), svc.score(x_test, y_test)

# Visualize decision region
plot_decision_regions(x.to_numpy(), y.to_numpy(),
                      clf = svc).

plt.show()
```

6. Support Vector Machines (Regression)

↳ Support Vector Regression (SVR) is a regression technique that uses Support Vector Machines (SVM) for modeling and predicting continous outcomes.



Working Practically

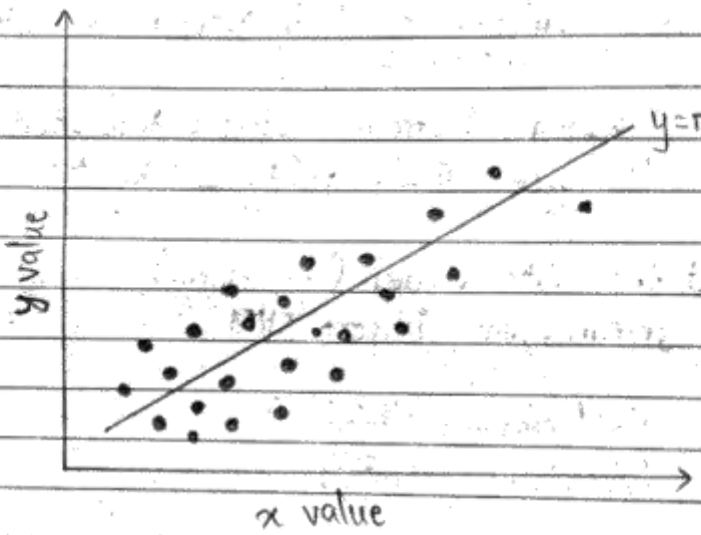
```
import pandas as pd.
import matplotlib.pyplot as plt
import seaborn as sns

dataset = pd.read_csv("Maths.csv")
dataset.head(3)

# Check for null values
dataset.isnull().sum()

# Drop all the null values
dataset.dropna(inplace = True)

# Visualize data
plt.figure(figsize = (5,5))
sns.scatterplot(x: "Number 3", y = "Total Sum", data =
                dataset)
plt.show()
```

```
# Seperate required dependent and independent data.
x = dataset[["Number3"]]
y = dataset["Total Sum"]

# Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y,
        test_size = 0.2, random_state = 42)

# Build the SVM model (Regression)
from sklearn.svm import SVR

svr = SVR(kernel = "linear")
svr.fit(x_train, y_train)

svr.score(x_test, y_test)

# Check if the model is overfitted
svr.score(x_train, y_train), svr.score(x_test, y_test)

# Visualize the prediction line.
sns.scatterplot(x = "Number3", y = "Total Sum", data =
                dataset)
plt.plot(dataset["Number3"], svr.predict(x), color = "red")
plt.show()
```

7. Hyperparameter Tuning

Model Parameters

- Model parameters are configuration variables that are internal to the model, and a model learns them on its own.



- From diagram, m and c are model parameters.

Model Hyperparameters

- Hyperparameters are these parameters that are explicitly defined by the user to control the learning process.
- The best value can be determined either by the rule of thumb or by trial and error.
- Example: DecisionTreeClassifier (max_depth = 3)

Hyperparameter Tuning

Models can have many hyperparameters and finding the best combination of parameters can be treated as a search problem.

The two best strategies for hyperparameter tuning are:

1. GridSearchCV
2. RandomizedSearchCV

1. **GridSearchCV :** GridSearchCV is a technique to search through the best parameter values from the given set of the grid of parameters.
It is slow when you have a large number of hyperparameters or huge dataset.

2. **RandomizedSearchCV :** It goes through only a fixed number of hyperparameter settings.
It moves within the grid in random fashion to find the best set of hyperparameters.
It can miss the best hyperparameters.

Working Practically:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

dataset = pd.read_csv("Maths.csv")
```

```
dataset.head(3)

# Check for null values
dataset.isnull().sum()

# Drop all null values
dataset.dropna(inplace = True)

# Visualize data
sns.scatterplot(x= dataset["Number 2", y= dataset["
                    Number3"], data = dataset)
plt.show()

# Select the required dependent and independent data
x = dataset[["Number 2"]]
y = dataset ["Number 3"]

# Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x,
        y, test_size = 0.2, random_state = 42)

# Build model
from sklearn.tree import DecisionTreeRegressor

dtr = DecisionTreeRegressor()
dtr.fit(x_train, y_train)

# Check if the model is overfitted
dtr.score(x_train, y_train), dtr.score(x_test, y_test)
```

```
# Yes it is overfitted so we need to do hyperparameter
# tuning to make it more accurate.

# Create a dictionary with the hyperparameters you
# want to tune
hyperparameters_to_tune = {
    "criterion": ["squared_error", "friedman_mse",
                  "absolute_error", "poisson"],
    "splitter": ["best", "random"],
    "max_depth": [i for i in range(2, 20)],
}
```

Using GridSearchCV :

```
from sklearn.model_selection import GridSearchCV

gdcv = GridSearchCV(DecisionTreeRegressor(), param_grid =
                    hyperparameters_to_tune)
gdcv.fit(x_train, y_train)

# Check best score
gdcv.best_score_

# Check the best hyperparameters value
gdcv.best_params_

# Train the model using best values of hyperparameters
dtr = DecisionTreeRegressor(criterion = "friedman_mse",
                            max_depth = 5, splitter = "random")
dtr.fit(x_train, y_train)
```

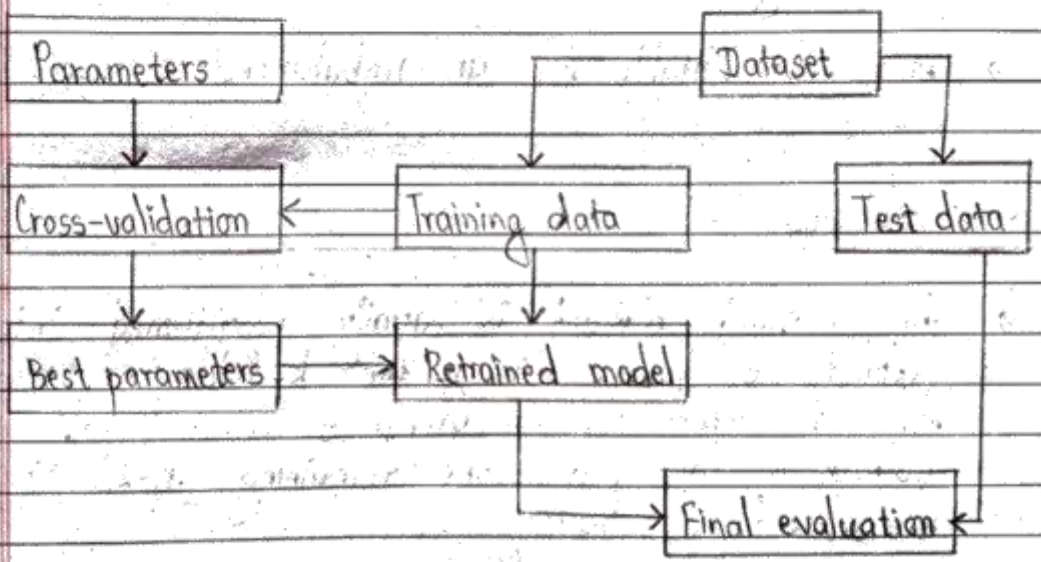
```
# Check if the model is overfitted.
dtr.score (x_train, y_train) , dtr.score (x_test , y_test)
# No this is not overfitted.
```

Using RandomizedSearchCV:

```
from sklearn.model_selection import RandomizedSearchCV

rdcv = RandomizedSearchCV ( DecisionTreeRegressor (),
                param_distributions = hyperparameters_to_tune,
                n_iter = 20 )
rdcv.fit (x_train , y_train)

# Check best score.
rdcv.best_score_

# Check the best hyperparameters value
rdcv.best_params_

# Train the model using best values of hyperparameters
dtr = DecisionTreeRegressor ( criteria = "friedman_mse",
        max_depth = 5 , splitter = "random" )
dtr.fit (x_train , y_train)

# Check if the model is overfitted
dtr.score (x_train , y_train) , dtr.score (x_test , y_test)
# No it is not overfitted.
```

8. Cross Validation

Cross-validation is a technique for validating the model efficiency by training it on the subset of input data and testing on previously unseen subset of input data.



Methods Used for Cross-Validation

- Leave p out cross-validation
- Leave one out cross-validation
- Holdout cross-validation
- Repeated random subsampling validation
- k-fold cross-validation
- Stratified k-fold cross validation
- Time series cross-validation
- Nested cross-validation.

• K-Fold Cross-Validation.

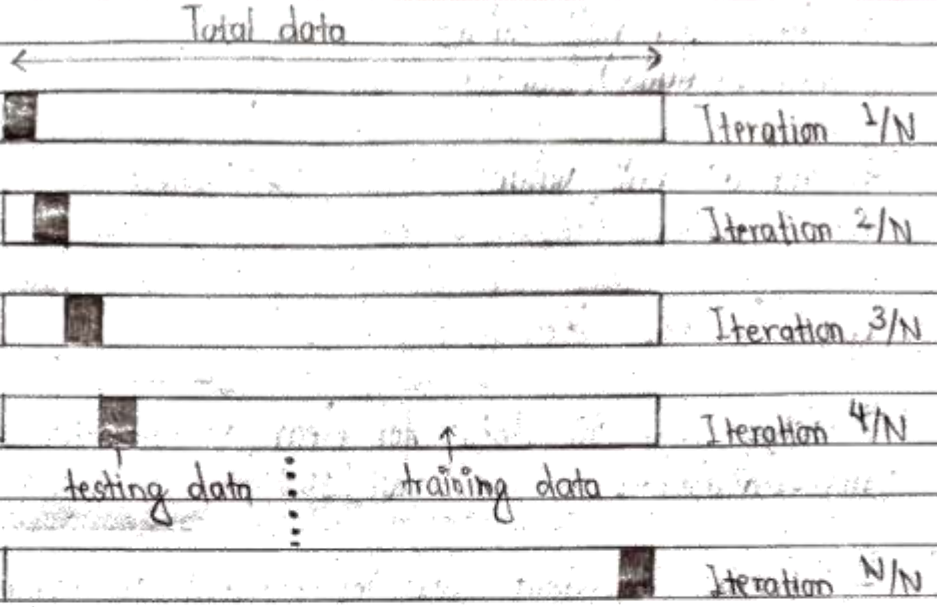
- The original dataset is equally partitioned into k subparts or folds. Out of the k-folds or groups, for each iteration, one group is selected as validation data, and the remaining (k-1) groups are selected as training data.
- It is not suitable for an imbalanced dataset.

• Stratified K-fold Cross-Validation

- The original dataset is equally partitioned into k subparts or folds. Out of the k-folds or groups, for each iteration, one group is selected as validation data, and the remaining (k-1) groups are selected as training data.
- Stratified k-fold cross-validation solve the problem of an imbalanced dataset.

• Leave-One-Out Cross-Validation

- Leave-one-out cross-validation (LOOCV) is an exhaustive cross-validation technique. It is a category of Lp OCV with the case of $p=1$.
- If the dataset is huge, it takes a lot time to train the model.



Leave P-Out Cross Validation

Leave p-out cross validation (LpOCV) is an exhaustive cross validation technique, that involves using p-observation as validation data and remaining data is used to train the model. This is repeated in all ways to cut the original sample on a validation set of p observations and a training set.

Working Practically.

```
import pandas as pd

dataset = pd.read_csv("Maths.csv")
dataset.head(3)
```

```
# Check for null values
dataset.isnull().sum()

# Drop all null values
dataset.dropna(inplace=True)
```

Check Validation Methods :

```
# Taking only 10 datas for clear visualization
new_dataset = dataset.head(10)

# Seperate dependent and independent variables
new_x = new_dataset.iloc[:, :-1]
new_y = new_dataset["Total Sum"]

# Check the validation methods in new_dataset
from sklearn.model_selection import LeaveOneOut,
                LeavePOut, KFold, StratifiedKFold

# LeaveOneOut
loo = LeaveOneOut()
for train, test in loo.split(new_x, new_y):
    print(train, test)

# LeavePOut
lpo = LeavePOut(p=2)
for train, test in lpo.split(new_x, new_y):
    print(train, test)

# KFold
kf = KFold(n_splits=5)
```

```
for train, test in kf.split(new_x, new_y):
    print(train, test)

# StratifiedKFold
# skf = StratifiedKFold(n_splits=5)
# for train, test in skf.split(new_x, new_y):
#     print(train, test)
''' StratifiedKFold works only on classification analysis '''
```

Check Range of Accuracy :

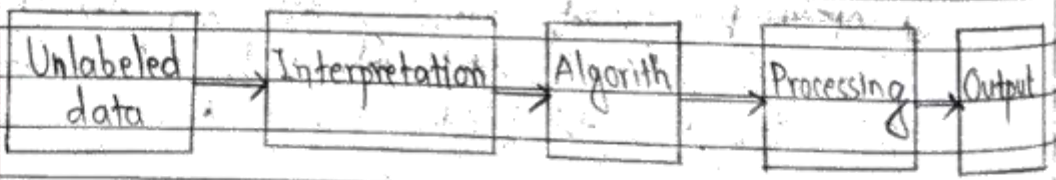
```
# Seperate dependent and Independent variables
x = dataset[["Number 1"]]
y = dataset["Total Sum"]

# Check how much accuracy can our dataset give.
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score

cross_val_score(LinearRegression(), x, y, cv=5)

# Sort the cross_val_score
cvs = cross_val_score(LinearRegression(), x, y, cv=5)
cvs.sort()
cvs*100  # Display in percentage.
```

7. Unsupervised Learning in ML

1. Introduction

- Unsupervised learning is a type of machine learning that learns from unlabeled data.
- This means that the data doesn't have any pre-existing labels or categories.
- The goal of unsupervised learning is to discover patterns and relationships in the data without any explicit guidance.

Remember: labeled data → has input and output
unlabeled data → only input no output.



Types of Unsupervised Learning

1. **Clustring:** Clustring is an unsupervised machine learning technique used to group similar data points into clusters based on their characteristics.
2. **Association:** Association rule learning is another unsupervised learning technique used to discover interesting relationships between variables in large datasets.

Popular Unsupervised Learning Algorithms.

- K-means clustering
- Hierarchal clustering
- DBSCAN clustering
- Apriori algorithm
- Principle component analysis.

2. K-Means Clustering

↳ K-means clustring is an unsupervised learning algorithm, which groups the unlabeled dataset into different clusters.

↳ K defines the number of pre-defined clusters that need to be created in the process.

Elbow Method.

↳ The elbow method is one of the most popular ways to find the optimal number of clusters.

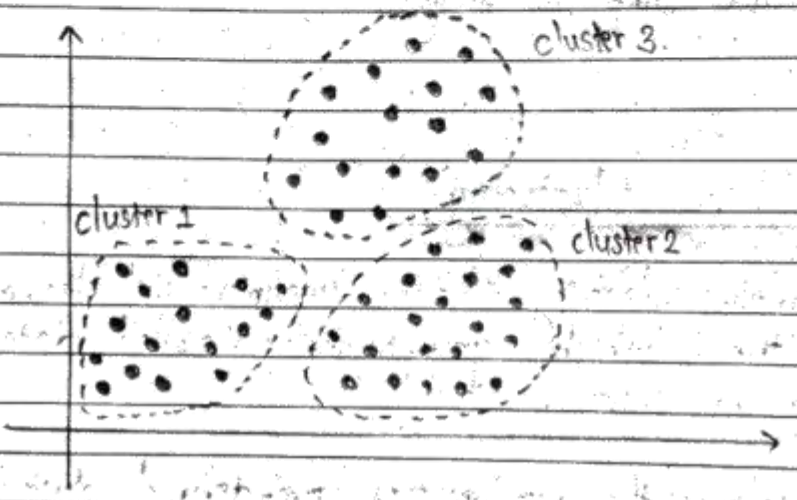
↳ This method uses the concept of WCSS value. WCSS stands for Within Cluster Sum of Squares, which defines the total variations within a cluster.

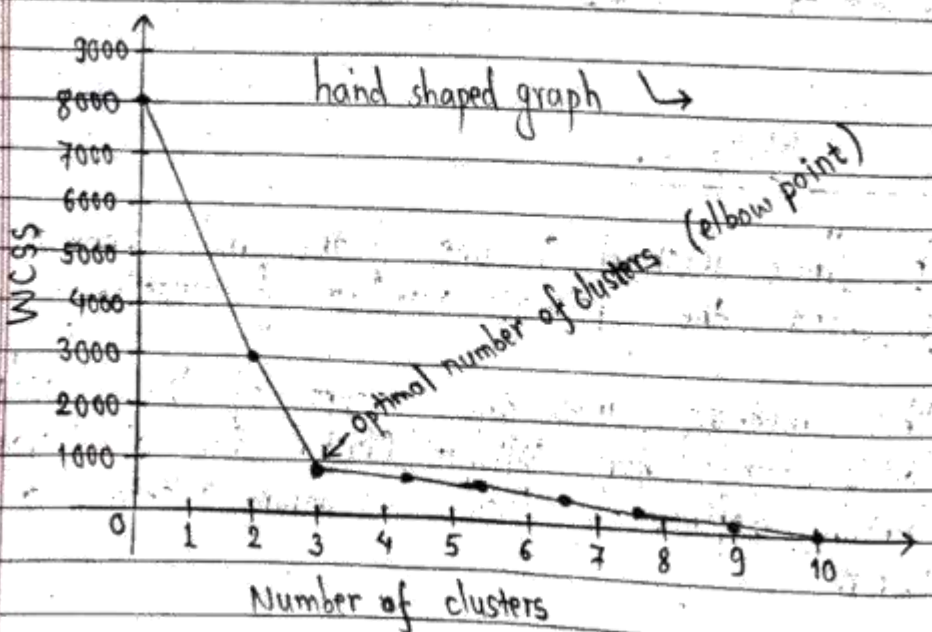
↳ Mathematically,

WCSS = $\sum_{p_i \text{ in cluster1}}$ distance $(P_i C_1)^2$ +

$\sum_{p_i \text{ in cluster2}}$ distance $(P_i C_2)^2$ +

$\sum_{p_i \text{ in cluster3}}$ distance $(P_i C_3)^2$ + ...





Working practically

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

dataset = pd.read_csv("IrisFlower.csv")
dataset.head(3)

# Our data is labeled
# For supervised learning we need unlabeled data
# So, lets remove Species column and make it
# unlabeled
dataset.drop(columns="Species", inplace=True)
dataset.head(3)

# Visualize the dataset
sns.pairplot(data=dataset)
plt.show()

# Find the best number of clusters
from sklearn.cluster import KMeans

wcs = []
for i in range(2,21):
km = KMeans(n_clusters=i, init="k-means++")
# k-means++ improves the initialization of cluster
#centroids, which helps in achieving better clustering
# results and faster convergence
km.fit(dataset)
wcss.append(km.inertia_) # Value of wcss
```

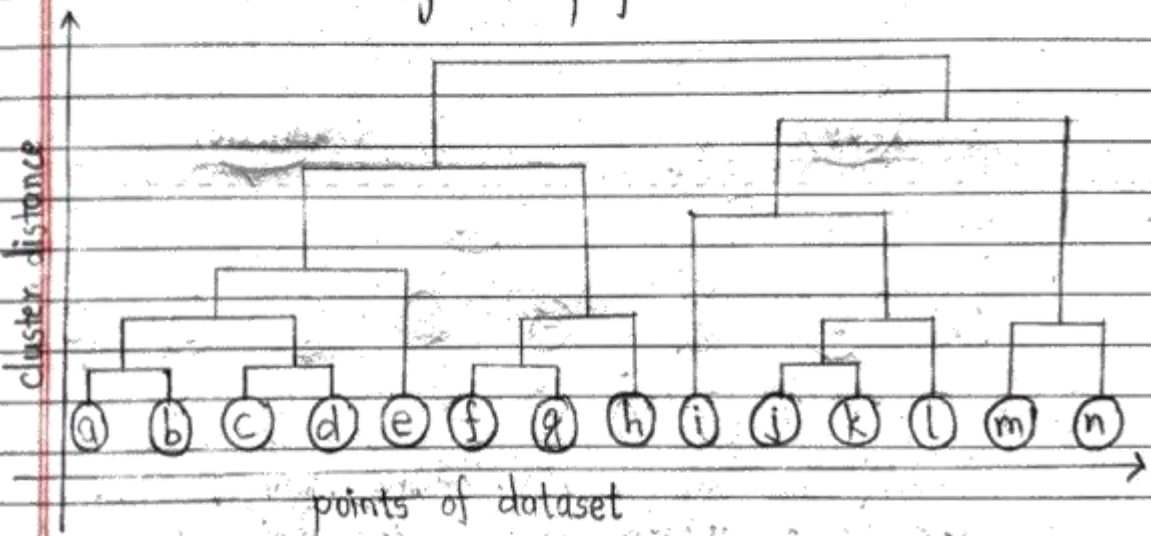
```
# Draw an elbow graph.
plt.figure (figsize = (10, 5))
plt.plot ([i for i in range (2, 21)], wcss,
                    marker = 'o')
plt.xlabel ("No of clusters")
plt.xticks ([i for i in range (2,21)]) # List of
# position at which to place the ticks
plt.ylabel ("WCSS")
plt.grid (axis = "x")
plt.show ()
"" 3 is found to be elbow point ""

# Make clusters with best value of k.
kmn = KMeans (n_clusters = 3)
kmn.fit_predict (dataset)

# Save the predicted value to dataset
dataset ["Predict"] = kmn.fit_predict (dataset)
dataset.head (3)

# Visualize clusters
sns.pairplot (dataset, hue = "Predict")
plt.show ()
```

3 Hierarchical Clustering

↳ It is used to group the unlabeled datasets into a cluster and also known as hierarchical cluster analysis or HCA.

- In this algorithm, we develop the hierarchy of clusters in the form of tree, and this tree-shaped structure is known as the dendrogram.

- The dendrogram : is a tree-like structure that is mainly used to store each step as a memory that the HC algorithm performs.



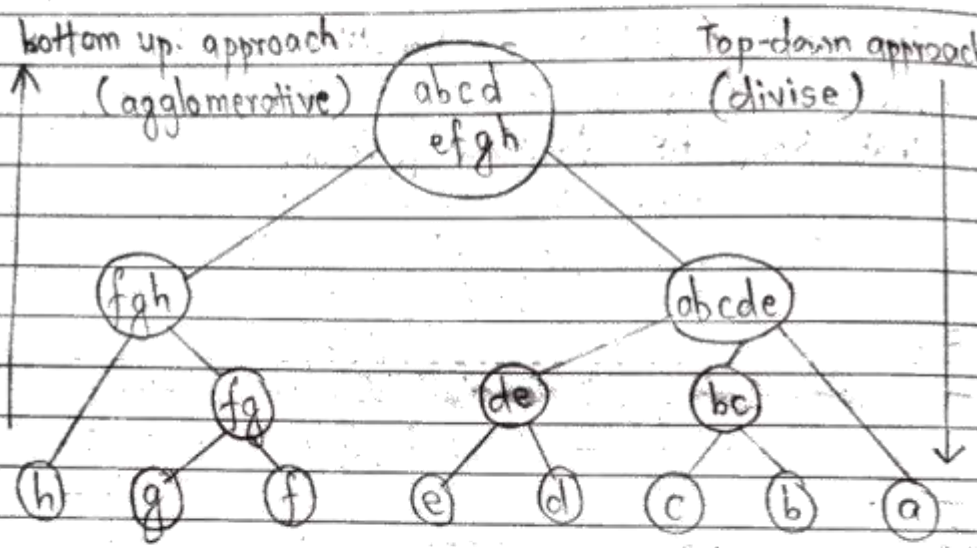
In dendogram plot, the Y-axis shows the Euclidean distances between the data points, and the x-axis shows all the data points of the given dataset.

Hierarchical Clustering Technique.

Hierarchical clustering technique has two approaches:

1. Agglomerative: Agglomerative is a bottom-up approach, in which the algorithm starts with taking all the data points as single cluster and merging them until one cluster is left.

2. **Divisive:** Divisive algorithm is the reverse of the agglomerative algorithm as it is a top-down approach.



Measure for the distance between two clusters:
The closest distance between the two clusters is crucial for the hierarchical clustering. There are various ways to calculate distance between two clusters, and these ways decide the rules for clustering. These measure are called Linkage methods.

- Single linkage (min)
- Complete linkage (max)
- Average linkage
- Centroid linkage.

Working practically

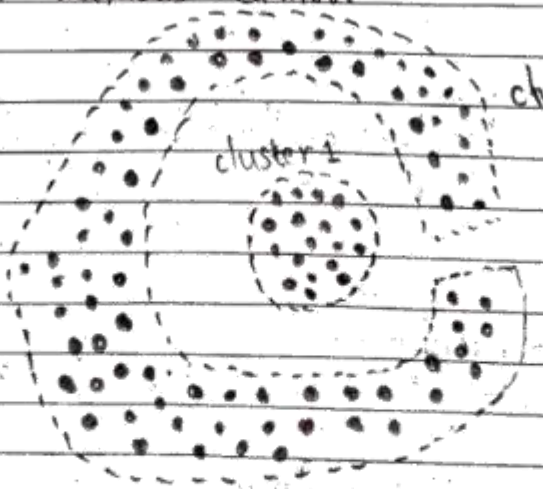
```
import pandas as pd
```

```
import matplotlib.pyplot as plt
import seaborn as sns

dataset = pd.read_csv("IrisFlower.csv")
dataset.head(3)

# We need a unlabeled dataset
# Converting a dataset to unlabeled
dataset.drop(columns="Species", inplace=True)
dataset.head(3)

# Visualize data
sns.pairplot(dataset)
plt.show()

# Create a dendrogram
import scipy.cluster.hierarchy as sc

plt.figure(figsize=(50, 25))
sc.dendrogram(sc.linkage(dataset, method="single",
                         metric="euclidean"))
plt.show()

# Make the clusters
from sklearn.cluster import AgglomerativeClustering

ac = AgglomerativeClustering(n_clusters=2, linkage=
                             "single")
ac.fit_predict(dataset)

# Save the predicted cluster value to dataset
```

```
dataset["Predict"] = ac.fit_predict(dataset)
dataset.head(3)

#Visualize the clusters
sns.pairplot(dataset, hue="Predict")
plt.show()
```

4. DBSCAN Clustring Algorithm

- Density-Based Spatial Clustering of Applications with Noise.
- The clusters found by DBSCAN can be any shape, which can deal with some special cases that other methods cannot.



Working Practically.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Create a dataset using make_moons
from sklearn.datasets import make_moons

x, y = make_moons (n_sample = 250, noise = 0.05)
x, y  # We just need x i.e. input to create
# unlabeled dataset

# Create a dataset using the data
df = {"Data 1": x [:, 0], "Data 2": x [:, 1], }
dataset = pd. DataFrame (df)
 dataset.head (3)

# Visualize your data
sns.scatterplot (x = "Data 1", y = "Data 2", data = dataset)
 plt.show ()

# Make clusters using DBSCAN Clustering Algorithm
 from sklearn. cluster import DBSCAN

dbscan = DBSCAN (eps = 0.2, min_samples = 5)
# eps is radius of circle and min_sample is
# minimum number of datapoints need to be in
# circle for continuing clustering
dbscan.fit_predict (dataset)

# Add prediction values to dataset
dataset ["Predict"] = dbscan. fit_predict (dataset)
```

```
dataset.head(3)

#Visualize clusters
sns.scatterplot(x="Data1", y="Data2", data=
                dataset, hue="Predict")
plt.show()
```

5. Silhouette Score

- Silhouette refers to a method of interpretation and validation of consistency within clusters of data.
- Silhouette Coefficient or silhouette score is a metric used to calculate the goodness of a clustering technique.
- Its value ranges from -1 to 1.
- Mathematically,

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}, \text{ if } |C_i| > 1$$

and

$$s(i) = 0, \text{ if } |C_i| = 1$$

The given mathematical formula can also be written as:

$$s(i) \begin{cases} 1-[a(i)/b(i)], & \text{if } a(i) < b(i) \\ 0 & \text{if } a(i) = b(i) \\ [b(i)/a(i)]-1 & \text{if } a(i) > b(i) \end{cases}$$

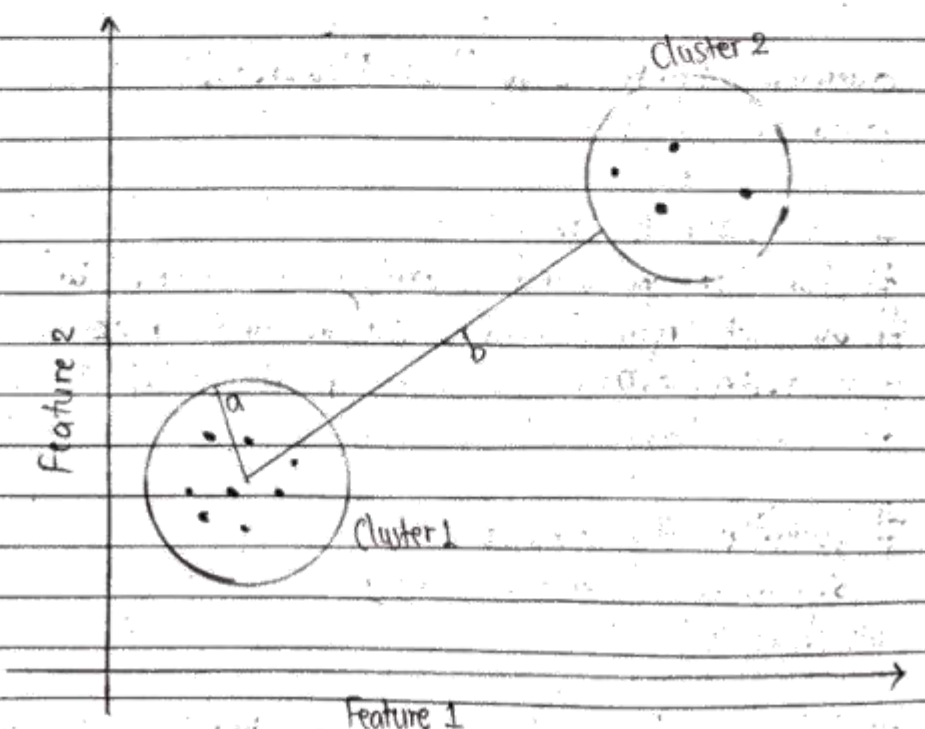
From above defination it is clear that:

$$-1 \leq s(i) \leq 1$$

Here:

$$a(i) = \frac{1}{|C_I|-1} \sum_{\substack{j \in C_I, \\ i \neq j}} d(i,j)$$

$$b(i) = \min_{J \neq I} \frac{1}{C_J} \sum_{j \in C_J} d(i,j)$$



a(i): the average distance from the datapoint (i) to all other points on same cluster.
b(i): the smallest average distance from the data point (i) to all other clusters that (i) is not a part of.

Remember: a(i) and b(i) are similar as a_i and b_i

Working Practically

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans

dataset = pd.read_csv("IrisFlower.csv")
dataset.head(3)

#Our data is labeled
# For unsupervised learning we need unlabeled data
# So lets remove species column and make it unlabeled
dataset.drop(columns="Species", inplace=True)
dataset.head(3)

# Visualize the dataset
sns.pairplot(data=dataset)
plt.show()

# Silhouette score to find best number of clusters
from sklearn.metrics import silhouette_score
```

```
# Find best number of clusters
ss = []
for i in range (2, 21):
    km = KMeans (n_clusters = i)
    km.fit (dataset)
    ss.append (silhouette_score (dataset , label =
                km.labels_ ))

# Create a graph between ss and number of clusters
no_of_clusters = [i for i in range (2, 21)]
plt.plot (no_of_clusters , ss, marker = "o")
plt.xlabel ("No. of clusters")
plt.xticks (no_of_clusters)
plt.ylabel ("Silhouette score")
plt.grid (axis = "x")
plt.show()
"""The best number of clusters is found to be 2"""

# Train model using best number of clusters.
km = KMeans (n_clusters = 2)
km.fit (dataset)

# Add the predict value to dataset for visualization
dataset ["Predict"] = km.predict (dataset)
dataset.head (3)

# Visualization of clusters
sns.pairplot (dataset , hue = "Predict")
plt.show()
```

6. Association Rule Learning

- Association rule learning is a machine learning technique used to discover interesting relationships or patterns in large datasets.
- It is often applied to transactional data, where items are bought or used together.
- Association rule learning works on the concept of if and else statement, such as if A then B. If element is called antecedent, and then statement is called as consequent
- Association rule learning work on the basis of: Support, Confidence and Lift

Support

- The first step for us and algorithm is to find frequently bought items.
- It is a straight forward calculation that is based on frequency.
- Mathematically,

  Support (A) = Transactions (A) / Total transactions

Confidence:

- We have identified frequently bought items let's calculate confidence.
- This will tell us how confident (based on our data) we can be that an item will be purchased, given that another item has been purchased.
- Mathematically,

$$\text{conf}(X \Rightarrow Y) = P(Y/X) = \frac{\text{supp}(X \cap Y)}{\text{supp}(X)}$$

$$= \frac{\text{number of transactions containing X and Y}}{\text{number of transactions containing X}}$$

Lift

- Given that different items are bought at different frequencies (strong association)
- Mathematically,

$$\text{lift}(X \Rightarrow Y) = \frac{\text{supp}(X \cap Y)}{\text{supp}(X) * \text{supp}(Y)}$$

- lift > 1 : means that the two items are more likely to be bought together.
  lift < 1 : means that the two items are more

likely to be bought seperately.
lift = 1 : means that there is no association between the two items.

Types of Association Rule Learning

- Apriori
- Eclat
- F-P Growth Algorithm

Applications of Association Rule Learning

- Market basket analysis
- Medical diagnosis
- Protein sequence

7. Apriori Algorithm

↳ The apriori algorithm is a fundamental algorithm in data mining used for mining frequent itemsets and discovering association rules.

Steps of Apriori Algorithm

1. Generate Candidate Itemsets : Start with single items and generate candidate itemsets of length (k) from frequent itemsets of length (k-1).

2. Prune Infrequent Itemsets: Remove candidate itemsets that don't meet the minimum support threshold.

3. Repeat: Repeat the process until no more candidate itemsets can be generated.

Working Practically

```
import pandas as pd

dataset = pd.read_csv("Groceries.csv")
dataset.head(3)

# Check for null values
dataset.isnull().sum()

# Create a list over lists by removing the null values
market = []
for i in range(0, dataset.shape[0]): # Till all row
# finish
        customer = []
        for j in dataset.columns : # Till all columns
            if type(dataset[j][i]) is str:
                customer.append(dataset[j][i])
        market.append(customer)
# Final list
market

# Identify most frequently bought products
import collections
```

```
# Convert the 2D list to 1D list
l = []
for i in market:
    for j in i:
        l.append(j)
# Display list
l
```

```
# Count total number of repetitions
collections.Counter(l)  # Takes 1D list
```

```
# Create a dataset for observation
item_count = collections.Counter(l)
df = {"Item Name": item_count.keys(), "Repetition":
      item_count.values()}
dataframe = pd.DataFrame(df).sort_values(by = ["
            Repetition"], ascending = False]
dataframe
```

```
# Select max number of rows to display
pd.set_option("display.max_rows", 200)
dataframe.
```

```
# Create a confusion matrix for Apriori Algorithm
from mlxtend.preprocessing import TransactionEncoder

tr = TransactionEncoder()
tr.fit(market)

tr.transform(market)
```

```
# Create a dataframe of confusion matrix
dframe = pd.DataFrame (tr.transform(market),
                       columns = tr.columns_)
dframe

# Use Apriori Algorithm
from mlxtend.frequent_patterns import apriori

apriori (dframe, min_support = 0.05, use_colnames = True,
         max_len = 3). sort_values (by = "support",
         ascending = False)
```

8. Frequent Pattern Growth Algorithm

Frequent Pattern growth algorithm is a popular method for mining frequent itemsets in large datasets. It addresses some of the limitations of the Apriori algorithm by using a more efficient data structure called FP-Tree (Frequent Pattern Tree)

Steps of FP-Growth Algorithm

1. Frequent Pattern set
2. Ordered-item set
3. Ordered-item set and conditional frequent pattern tree is built.
4. Frequent pattern rules are used.

Working Practically

```
import pandas as pd

# Set max number of rows to display.
pd.set_option("display.max_rows", 200)

dataset = pd.read_csv("Groceries.csv")
dataset.head(3)

# Create a list over lists removing the null values
market = []
for i in range(0, dataset.shape[0]): # Till all rows
    customer = []
    for j in dataset.columns: # Till all columns
        if type(dataset[j][i]) is str:
            customer.append(dataset[j][i])
    market.append(customer)
# Final list
market

# Identify most frequently bought products
import collections

# Convert the 2D list to 1D list
l = []
for i in market:
    for j in i:
        l.append(j)
# Count total number of repetition
collections.Counter(l) # Takes 1D list
```

```
# Create a dataset for observation
item_count = collections.Counter(1)
df = {"Item Name" : item_count.keys(),
      "Repetition" : item_count.values()}
dataframe = pd.DataFrame(df).sort_values(by = [
      "Repetition"], ascending = False)
dataframe

# Create a confusion matrix (Data Encoding)
from mlxtend.preprocessing import TransactionEncoder

tr = TransactionEncoder()
tr.fit(market)

tr.transform(market)

# Create a dataframe of confusion matrix
dframe = pd.DataFrame(tr.transform(market), columns =
      tr.columns_)
dframe

# Use frequent pattern growth algorithm
from mlxtend.frequent_patterns import fpgrowth

fpgrowth(dframe, min_support = 0.05, use_colnames = True,
      max_len = 3).sort_values(by = "support",
      ascending = False)
```

8. Ensemble Learning in ML

1. Introduction

- The Ensemble methods in machine learning combine the insights obtained from multiple learning models to facilitate accurate and improved decisions.

- There are two main types of ensemble learning:
1. Bagging (Bootstrap Aggregating)
2. Boosting.

Ensemble learning Techniques

Basic Ensemble Techniques:
- Max voting
- Averaging
- Weighted average

Advanced Ensemble Techniques:
- Stacking
- Blending
- Bagging
- Boosting.

Algorithms Based on Bagging:
- Bagging meta-estimator
- Random forest

Algorithms Based on Boosting:
- AdaBoost

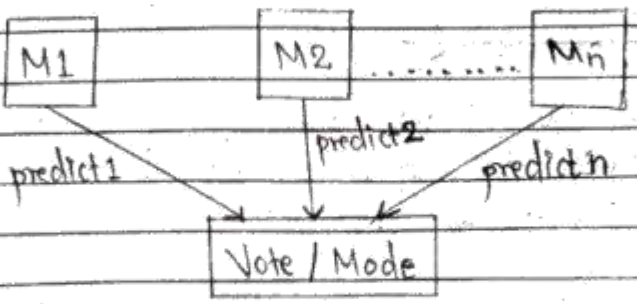
- GBM
- xGBM
- Light GBM
- Cat Boost

2. Basic Ensemble Techniques

1. Max Voting

- The max voting method is generally for classification problems. In this technique, multiple models are used to make predictions for each data point.

- The predictions by each model are considerd as a 'vote'. The predictions which we get from majority of the models are used as the final prediction.

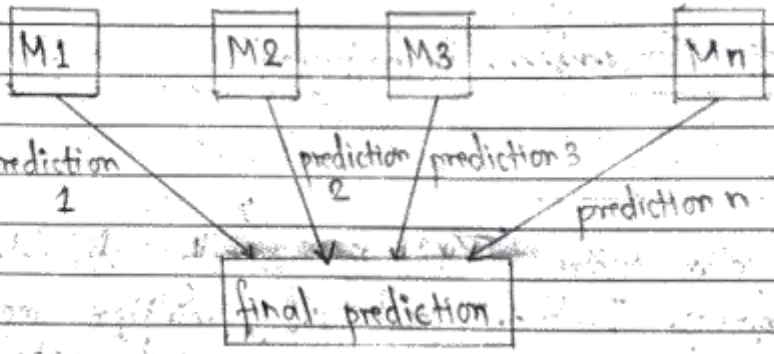


2. Averaging and Weighted Average Voting

- Take an average of predictions from all the models

and use it to make the final prediction.

6 Averaging can be used for making predictions in regression problems or while calculating probabilities for classification problems.



6 Weighted means we have assigned the weight to models.

Working Practically

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Voting Classifier:

```
# Create a dataset using make_moons
from sklearn.datasets import make_moons

# Get data for dataset
```

```
x, y = make_moons(n_samples = 1000, noise = 0.2)
# Here x is independent and y is dependent data

# Create a dataframe using x and y for tabular
# visualization
df = {"x1": x[:,0], "x2": x[:,1], "y": y}
dataset = pd.DataFrame(df)
dataset.head(3)

# Visualize data
sns.scatterplot(x = "x1", y = "x2", data = dataset, hue =
                "y")
plt.show()

# Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y,
        test_size = 0.2, random_state = 42)

# Train different models
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB

# DecisionTree Classifier
dtc = DecisionTreeClassifier()
dtc.fit(x_train, y_train)
dtc.score(x_train, y_train); dtc.score(x_test, y_test)

# SVC
```

```
svc = SVC()
svc.fit(x_train, y_train)
svc.score(x_train, y_train), svc.score(x_test, y_test)

# Use Ensemble Learning (VotingClassifier)
from sklearn.ensemble import VotingClassifier

# Create a list of tuples with classifier names and
# instances (estimator)
l = [("dtc", DecisionTreeClassifier()),
     ("svc", SVC()),
     ("gnb", GaussianNB())]

# Create the Voting Classifier model.
vc = VotingClassifier(l)
# You can also use weightage in terms of list:
#  weights = [1, 2, 3]
vc.fit(x_train, y_train)

vc.score(x_train, y_train), vc.score(x_test, y_test)

# Lets create a dataframe for clear visualization
predicted = {
    "DecisionTreeClassifier": dtc.predict(x_test),
    "SVC": svc.predict(x_test),
    "GaussianNB": gnb.predict(x_test),
    "VotingClassifier": vc.predict(x_test),
}
# Create a dataframe
pd.DataFrame(predict)
# The most repeated result (mode) is final result
```

by Voting Classifier

Voting Regressor :

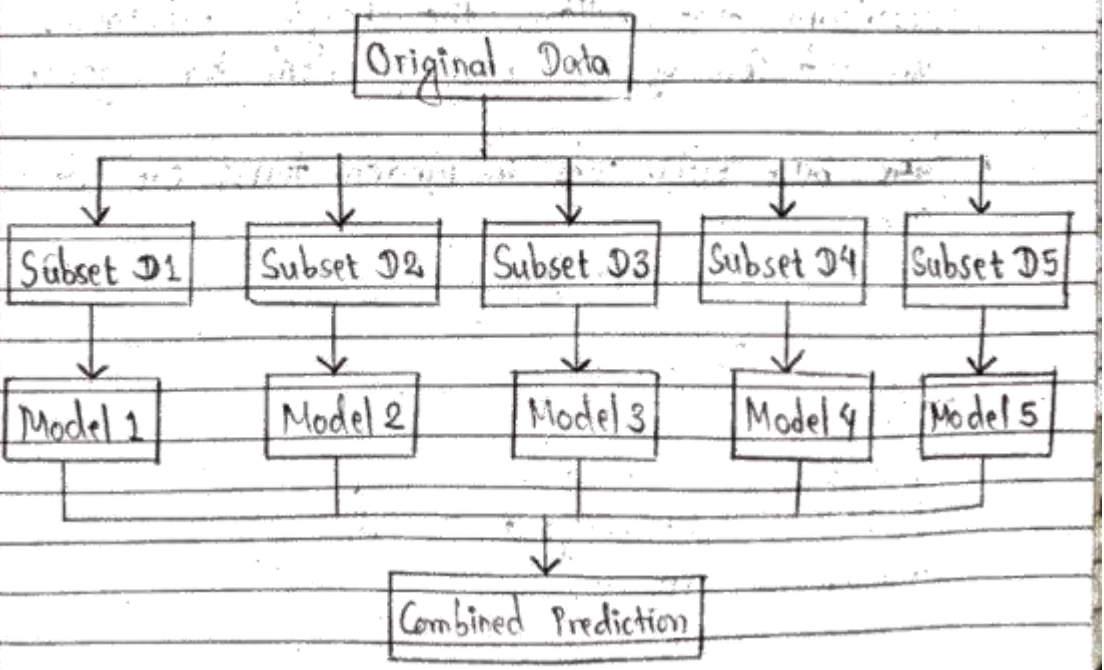
```
dataset = pd.read_csv("Maths.csv")
dataset.head(3)

# Check for null values
dataset.isnull().sum()

# Remove all the null values instead of filling (big
# dataset)
dataset.dropna(inplace=True)

# Select dependent and 'independent data
x = dataset[["Number1"]]
y = dataset["Total Sum"]

# Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x,
        y, test_size=0.2, random_state=42)

# Train different models
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR

# LinearRegression
lr = LinearRegression()
```

```
lr.fit (x_train, y_train)
lr.score (x_train, y_train) , lr.score (x_test, y_test)

# Decision Tree Regressor
dtr = DecisionTreeRegressor()
dtr.fit (x_train, y_train)
dtr.score (x_train, y_train) , dtr.score (x_test, y_test)

#SVR
svr = SVR()
svr.fit (x_train, y_train)
svr.score (x_train, y_train) , svr.score (x_test, y_test)

# Use Ensemble Learning (Voting Regressor)
from sklearn.ensemble import VotingRegressor

# Create a list of tuples with classifier names and
# Instances (estimators)
l = [("lr", LinearRegression()), ("dtr",
    DecisionTreeRegressor()), ("svr", SVR())]

vr = VotingRegressor (l, weights=[2, 3, 4])
vr.fit (x_train, y_train)

vr.score (x_train, y_train) , vr.score (x_test, y_test)

# Now lets see how VotingRegressor works by
# Comparision on dataset.
df = {
    "LinearRegression": lr.predict (x_test),
    "DecisionTreeRegressor": dtr.predict (x_test),
```

```
    "SVR" : svr.predict(x_test)
    "VotingRegressor" : vr.predict(x_test),
}
# Create a dataframe
pd.DataFrame(df).head(3)
# VotingRegressor gives average of other models
```

3. Bagging

- Bagging (or Bootstrap Aggregating) technique uses these subsets (bags) to get a fair idea of the distribution (complete set).
- The size of subsets created for bagging may be less than the original set.



- Bagging algorithms are:
1. Bagging meta-estimator
2. Random forest

1. Bagging Meta-Estimator

- Bagging meta-estimator is an ensembling algorithm that can be used for both classification (Bagging Classifier) and regression (Bagging Regressor) problems.
- It follows the typical bagging technique to make predictions.

2. Random Forest

- Random forest is another ensemble machine learning algorithm that follows the bagging technique. It is an extension of the bagging estimator algorithm.
- The base estimators in random forest are decision trees.

Working Practically

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Classification Analysis:

```
# Create a dataset using make_moons
from sklearn.datasets import make_moons

# Get data for dataset
x, y = make_moons (n_samples = 1000, noise = 0.2)
# Here x is independent and y is dependent data

# Create a dataframe using x and y for tabular
# visualization
df = {"x1": x[:,0], "x2": x[:,1], "y": y}
dataset = pd.DataFrame (df)
dataset.head (8)

# Visualize data
sns.scatterplot (x="x1", y="x2", data = dataset,
                 hue = "y")
plt.show()

# Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split (x,
        y, test_size = 0.2, random_state = 42)

# Classification analysis using Bagging Meta-Estimator
# technique
from sklearn.ensamble import BaggingClassifier

# We need an estimator for BaggingClassifier
```

```
from sklearn.svm import SVC

# Check for the score of our estimator
svc = SVC()
svc.fit(x_train, y_train)
svc.score(x_train, y_train), svc.score(x_test, y_test)

# Create the BaggingClassifier model
bc = BaggingClassifier(estimator=SVC(), n_estimators=50)
bc.fit(x_train, y_train)

bc.score(x_train, y_train), bc.score(x_test, y_test)

# Classification analysis using Random Forest technique
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators=50)
rfc.fit(x_train, y_train)

rfc.score(x_train, y_train), rfc.score(x_test, y_test)
```

Regression Analysis:

```
dataset = pd.read_csv("Maths.csv")
dataset.head(3)

# Check for null values
dataset.isnull().sum()

# Remove all the null values
dataset.dropna(inplace=True)
```

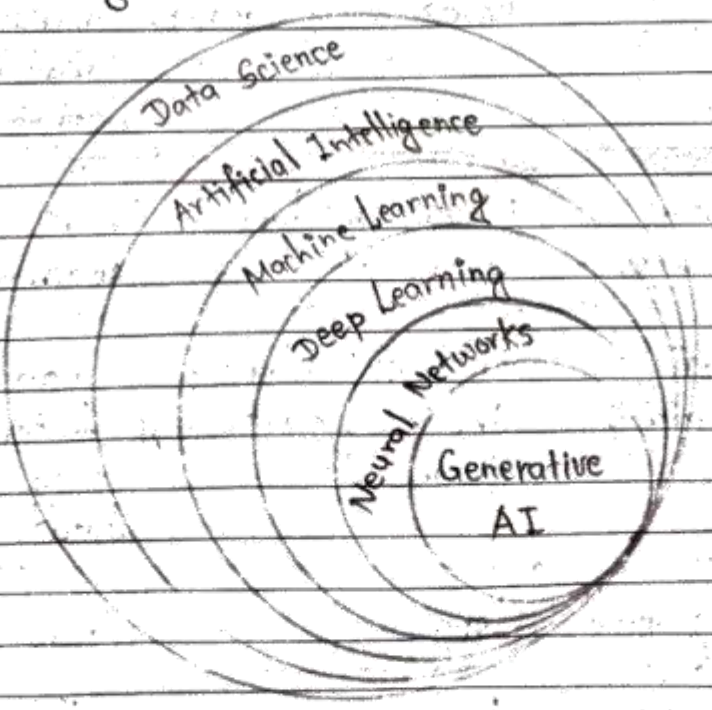
```
# Select dependent and independent data
x = dataset [["Number 3"]]
y = dataset ["Total Sum"]

# Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split (x,
        y, test_size = 0.2 , random_state = 42)

# Regression analysis using Bagging Meta-Estimator
# technique
from sklearn.ensemble import BaggingRegressor

# We need an estimator for BaggingRegressor
from sklearn.linear_model import LinearRegression

# Check for the score of our estimator.
lr = LinearRegression ()
lr.fit (x_train, y_train)
lr.score (x_train, y_train) , lr.score (x_test, y_test)

# Create the BaggingRegressor model
br = BaggingRegressor (estimator = LinearRegression () ,
                       n_estimators = 50)
br.fit (x_train , y_train)

br.score (x_train , y_train) , br.score (x_test, y_test)

# Regression analysis using Random Forest technique
from sklearn.ensemble import RandomForestRegressor
```

```
rfr = RandomForestRegressor(n_estimators=50)
rfr.fit(x_train, y_train)

rfr.score(x_train, y_train), rfr.score(x_test, y_test)
```

9. Deep Learning and AI

1. Deep Learning

- Deep learning is a collection of statistical techniques of machine learning feature hierarchies that are actually based on artificial neural networks.
- While the term deep learning is vague, it bases on the idea of mimicking the brain by building algorithms that resemble biological neurons' functionality in the brain.



Applications

- Computer Vision
- Natural - Language Processing (NLP)

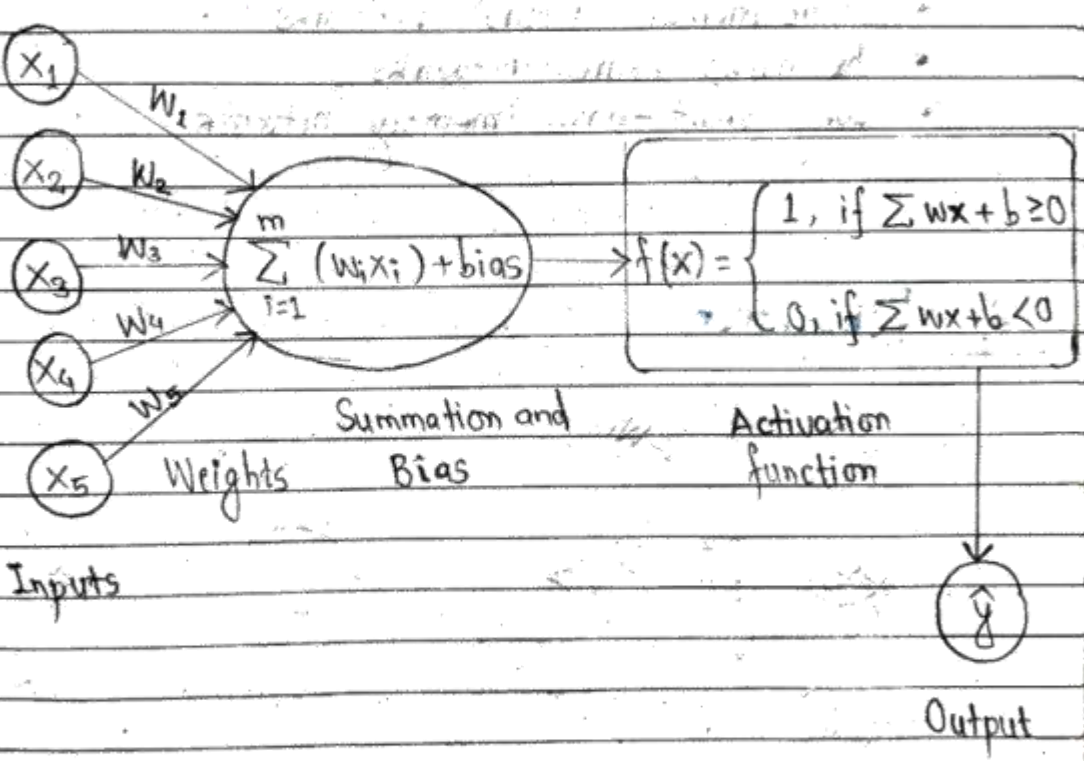
- Speech Recognization
- Health care
- Language translation
- Image detection
- Auto pilot cars

Deep Learning Vs. Machine Learning

	Deep Learning	Machine Learning
Data	Needs a big dataset.	Performs well with a small to a medium dataset.
Hardware requirements	Requires machines with GPU.	Works with low-end machines.
Engineering peculiarities	Needs to understand the basic functionality of the data.	Understands the features and how they represent the data
Training time	Long	Short
Processing time	A few hours or weeks.	A few seconds or hours.
Number of algorithms	Few	Many

Data interpetation	Difficult	Some ML algorithms are easy and some are hard.

Neural Network

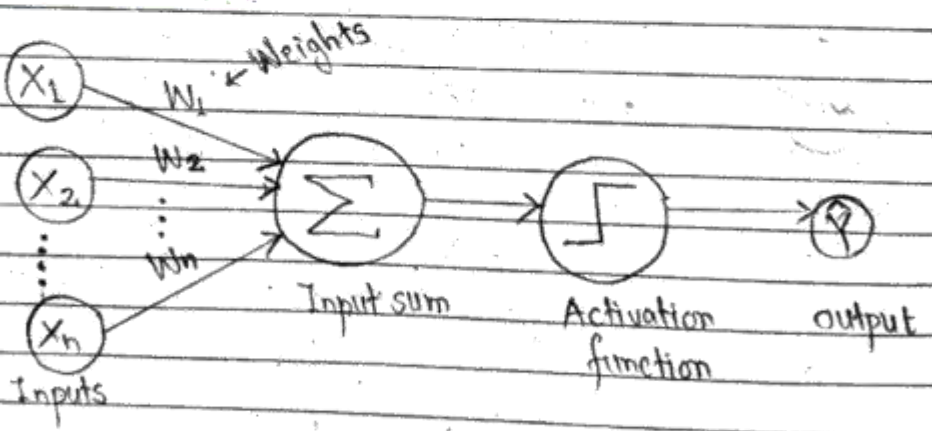


- A neural network is a type of machine learning model by the structure and function of human brain.
- It consists of interconnected units called neurons, which are organized into layers.
- Structure : Input layer → Hidden layer → Output layer
- Working : forward propagation and back - propagation

Types of Deep Learning Networks

- Perceptron (single layer)
- Feed forward networks
- Multi-layer perceptron (ANN)
- Radial based networks
- Convolutional neural networks
- Recurrent neural networks
- Long short-term memory network

2. Perceptron



A perceptron is one of the simplest type of artificial neural networks and serves as a fundamental building block for more complex neural network architectures

Working

1. Input features : The perceptron takes multiple input

features, each representing characteristics of the input data.

2. **Weights:** Each input feature is associated with a weight, which determines its influence on the output.

3. **Summation function:** The perceptron calculates the weighted sum of its input.

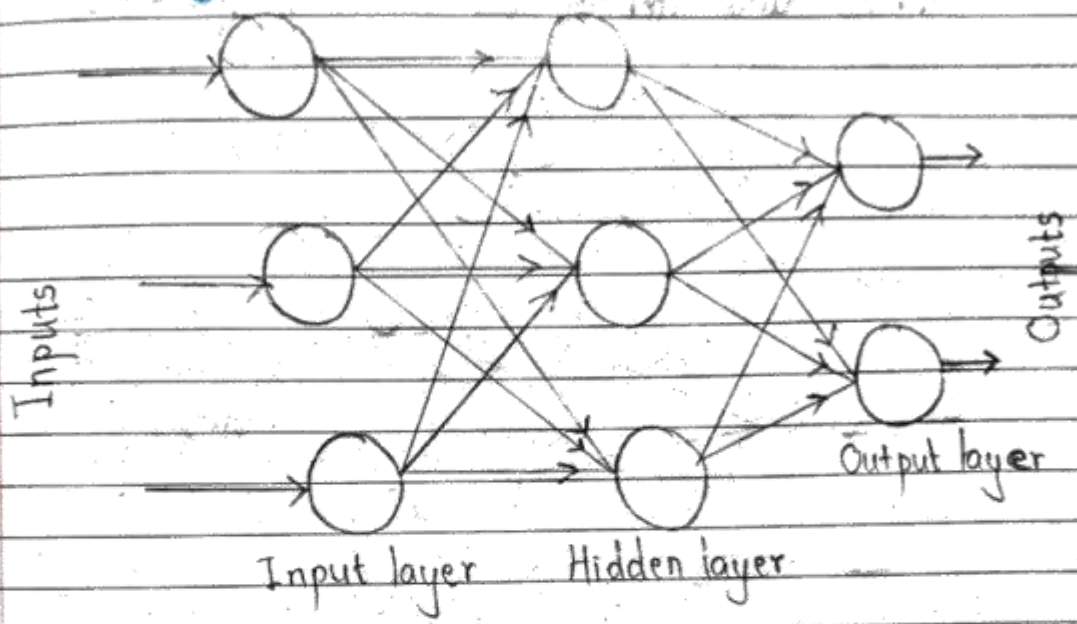
4. **Activation function:** The weighted sum is passed through an activation function (typically a step function) to produce the output. The output is usually binary (0 or 1)

Working Practically

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from mlxtend.plotting import plot_decision_regions

dataset = pd.read_csv("Suscription.csv")
dataset.head(3)

# Check for null values
dataset.isnull().sum() # No null values.

# Visualize data
plt.figure(figsize=(4,3))
```

```
sns.scatterplot(x="Age", y="Salary", data=dataset,
                hue="Purchase")
plt.show()

# Seperate input and output
x = dataset.iloc[:, :-1]
y = dataset["Purchase"]

# Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y,
        test_size = 0.2, random_state = 42)

# Perceptron model
from sklearn.linear_model import Perceptron

pr = Perceptron()
pr.fit(x_train, y_train)

pr.score(x_train, y_train), pr.score(x_test, y_test)

# Visualize decision region.
plot_decision_regions(x.to_numpy(), y.to_numpy(),
                      clf = pr)
plt.show()
"""This cannot provide high accuracy. So, Instead of
this multilayer perceptron is used"""
```

3. Multilayer Perceptron

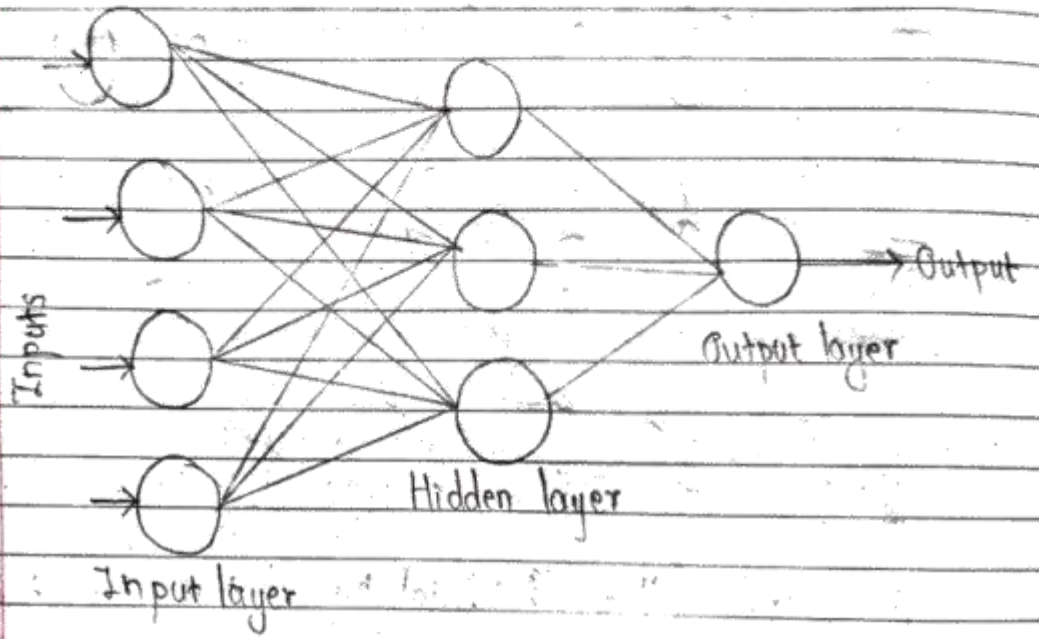


- It is also called Artificial Neural Networks (ANN)
- A multilayer perceptron (MLP) is a type of artificial neural network that consist of multiple layers of neurons, typically including an input layer, one or more hidden layers and one output layer.
- Working is based on forward propagation and back-propagation.

Forward Propagation

- Data is passed through the network from the input layer to the output layer.

- Each neuron computes a weighted sum of its inputs and applies an activation function.



Back - Propagation

- Backpropagation is a technique of ajustment of weights.

- The error between the predicted output and actual output is calculated. The error is propagated back through the network and weights are adjusted to minimize the error.

- Uses gradient descent technique (formula).

  i.e. $W_{new} = W_{old} - \lambda \left(\frac{dL}{dW} \right)$

  where λ is learning rate

Working Practically

```
import pandas as pd

dataset = pd.read_csv("ChurnModelling.csv")
dataset.head(3)

#Drop columns with object (string) type data
dataset = dataset.select_dtypes(exclude=["object"])
dataset.head(3)

#Check for null values
dataset.isnull().sum()  #No null values

#Seperate input and output data
input_data = dataset.iloc[:, :-1]
output_data = dataset.iloc[:,-1]

#Scaling the input data
from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
input_data = pd.DataFrame(ss.fit_transform(input_data),
                          columns=input_data.columns)

#Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(input_
    data, output_data, test_size=0.2, random_state=
    42)
```

```
# Create an artificial neural network
import tensorflow
import keras.layers import Dense
import keras.models import Sequential

ann = Sequential() # Object initialization

# Check the shape of training data
print(x_train.shape, y_train.shape) # Ensure shape are
# correct

# Build different layers.
ann.add(Dense(8, input_dim=x_train.shape[1],
              activation="relu")) # first hidden layer
# first parameter 8 is the number of nodes in the
# layer
ann.add(Dense(6, activation="relu")) # Second hidden
# layer
# Input dimension is needed only for first hidden layer
ann.add(Dense(4, activation="relu")) # Third hidden
# layer
ann.add(Dense(2, activation="relu")) # Fourth hidden
# layer
ann.add(Dense(1, activation="sigmoid")) # Output
# layer
# Sigmoid is used because our output is binary

# Compile the model
ann.compile(optimizer="adam", loss="binary_crossentropy"
            metrics=["accuracy"])
```

```
# Train the model
ann.fit (x_train, y_train, batch_size = 100, epochs = 50)
# Using batches helps in efficient computation and
# faster convergence.
# Batches allows the model to update it's weights
# more frequently than if it were using the entire
# dataset
# Multiple epochs are used to train the model,
# allowing it to learn and adjust its weights
# iteratively.

# Check the accuracy of our model.
from sklearn.metrics import accuracy_score

# We need prediction value of testing data.
ann.predict (x_test) # The data is not in binary form.

# Convert the data in binary form
prediction = ann.predict (x_test)
binary_prediction = []
for i in prediction:
        if i[0] > 0.5:
                binary_prediction.append (1)
        else:
                binary_prediction.append (0)
# View prediction
binary_prediction

accuracy_score (y_test, binary_prediction) * 100
# Accuracy of testing data
```

```
# For accuracy of training data
# We need prediction value of training data in
# binary format
prediction = ann.predict(x_train)
binary_prediction = []
for i in prediction:
    if i[0] > 0.5:
        binary_prediction.append(1)
    else:
        binary_prediction.append(0)
# View prediction
binary_prediction

accuracy_score(y_train, binary_prediction) * 100
# Accuracy of training data.

# To make the new predictions
import numpy as np

new_data = ss.fit_transform((([[1, 15634602, 619, 42,
                2, 0.00, 1, 1, 1, 101348.88]]))
new_prediction = ann.predict(new_data)
if new_prediction > 0.5:
    binary_form = 1
else:
    binary_form = 0
# View output
binary_form
```

Date Page 210

4. Activation Functions

- An activation function decides whether a neuron should be activated or not. This means that it will decide whether the neuron's input to the network is important or not in the process of prediction using simpler mathematical operations.

- The activation function is categorized into three main parts:

1. Binary step function
2. Linear activation function
3. Non-linear activation function.

1. Binary Step Function

- Binary step function depends on a threshold value that decides whether a neuron should be activated or not.

- Mathematically, binary step function is:

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$

Where,

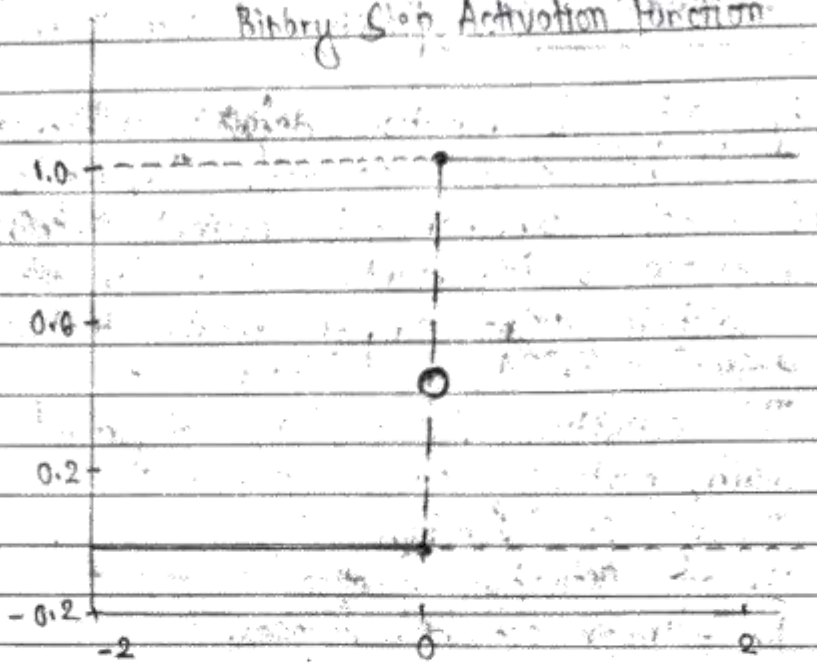
$$f(x) = \hat{y} = x_1 w_1 + x_2 w_2 + \dots + x_n w_n + b$$

where,

$\hat{y}$ is predicted output

$x_1, x_2, \dots x_n$ are input values

$w_1, w_2, \dots w_n$ are weightages.



2. Linear Activation Function

- The output of the functions is not restricted in between any range.
- Its range is specified from $-\infty$ to ∞
- Mathematically, linear activation function is:

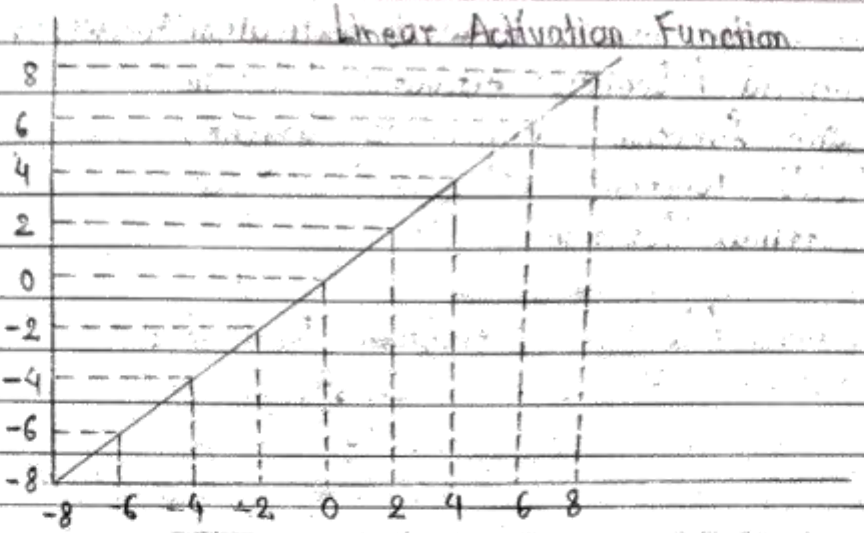
$$f(x) = x$$

i.e.

$$\hat{y} = x$$

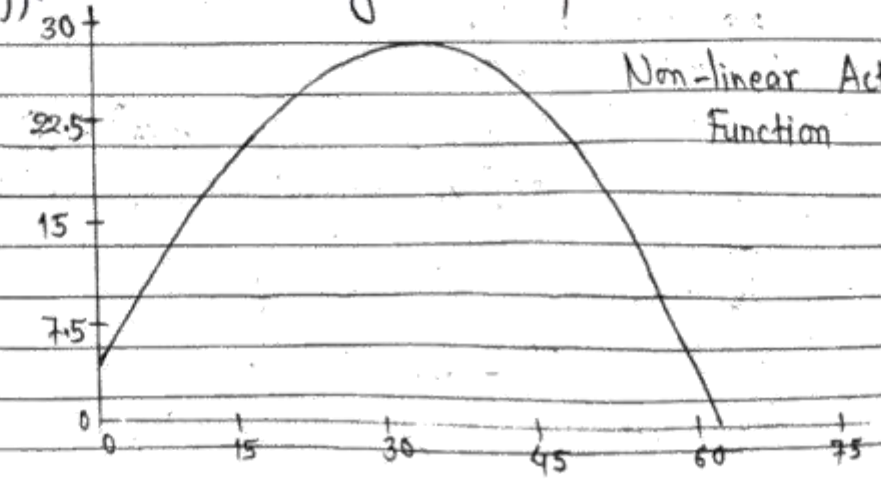
where,

$\hat{y}$ is predicted output and
x is input.



3. Non-Linear Activation Functions

- These are one of the mostly widely used activation functions.
- It helps the model in generalizing and adapting any sort of data in order to perform correct differentiation among the output.

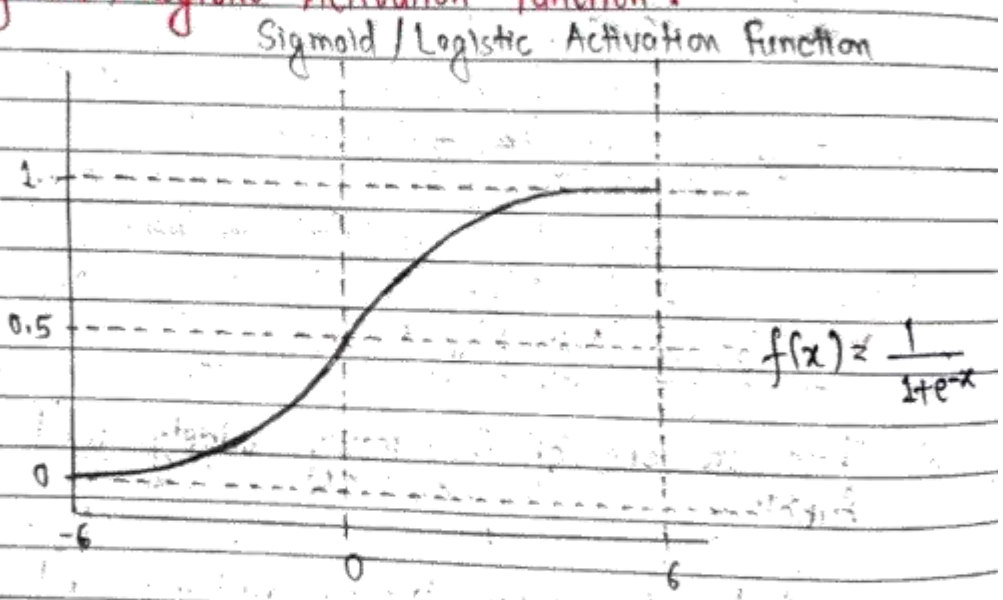


Date
Page 215

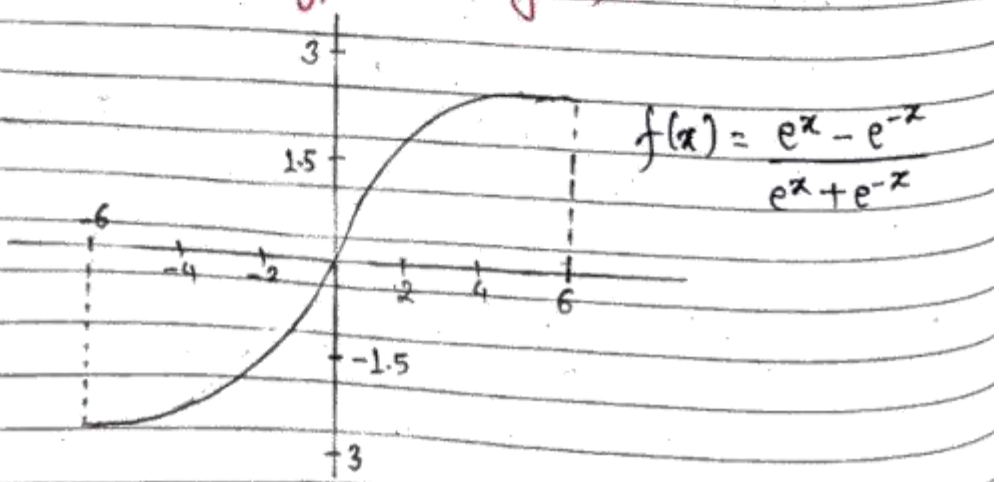
Non-linear Neural Network Activation Functions:

i. Sigmoid / Logistic Activation Function
ii. Tanh function (Hyperbolic Tangent)
iii. ReLU Function
iv. Softmax function

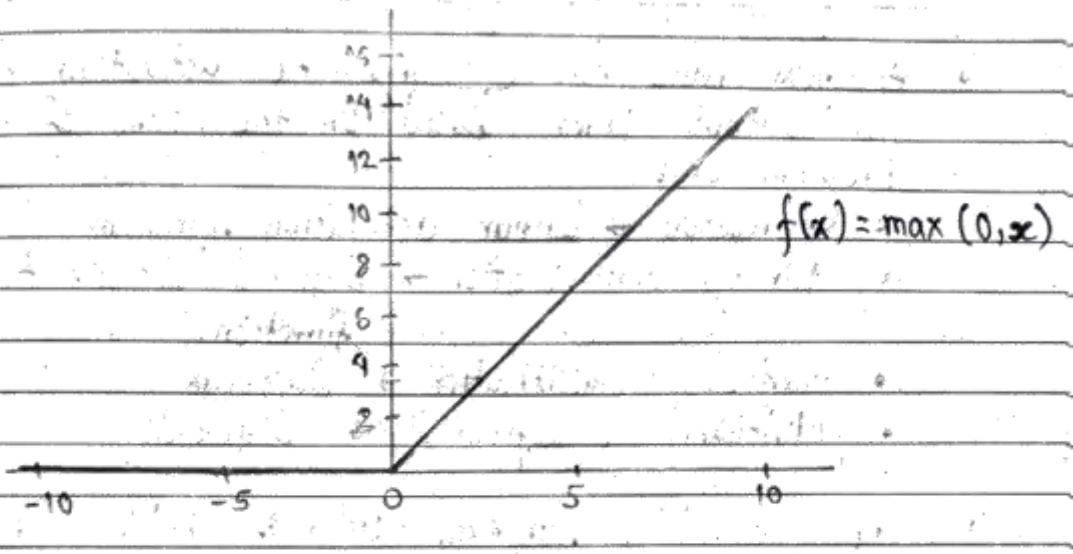
i. Sigmoid / Logistic Activation Function:



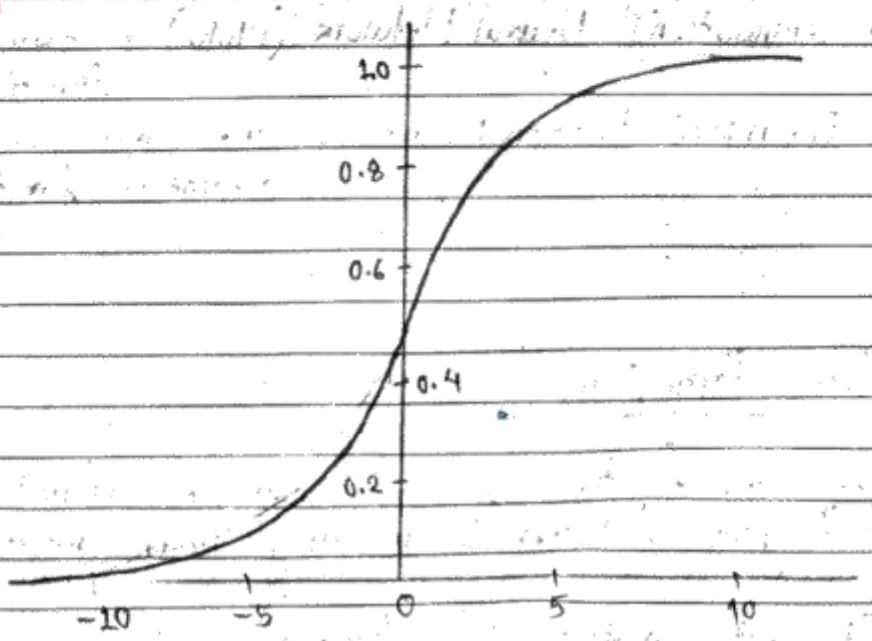
ii. Tanh Function (Hyperbolic Tangent):



iii. ReLU Function:



iv. Softmax Function:



$$\text{Softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

Rules for Choosing Right Activation Function

- A few rules for choosing the activation function for output layer based on the type of prediction problem are:
  - Regression → linear activation function
  - Binary classification → Sigmoid / Logistic Activation function.
  - Multiclass classification → Softmax
  - Multilabel classification → Sigmoid
- The activation function used in hidden layers is typically choosen based on the type of neural network architecture:
  - Convolutional Neural Network (CNN): ReLU activation function
  - Recurrent Neural Network: Tanh and/or Sigmoid activation function

5. Loss Functions

- The loss function is a method of evaluating how well your algorithm is modeling your dataset.
- It is the mathematical function of the parameters of the machine learning algorithm.
- Remember that cost function is average of loss function of all data in dataset.

Types of Loss Functions

- **Regression :**
  - MSE (Mean Squared Error)
  - MAE (Mean Absolute Error)
  - Hubber loss
- **Classification :**
  - Binary cross-entropy / log loss
  - Categorical cross-entropy
- **Auto Encoder :**
  - KL Divergence.
- **GAN :**
  - Discriminator loss
  - Minmax GAN loss
- **Object Detection :**
  - Focal loss
- **Word Embeddings :**
  - Triplet loss

Regression Loss

1. **Mean Squared Error / Squared loss / L2 loss :**

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (Y_i - \hat{Y}_i)^2$$

2. Mean Absolute Error / L1 loss:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|Y_i - \hat{Y}_i|$$

3. Huber Loss:

$$Huber = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}(y_i - \hat{y}_i)^2 \quad , |y_i - \hat{y}_i| \leq \delta$$

$$Huber = \frac{1}{n}\sum_{i=1}^{n}\delta\left(|y_i - \hat{y}_i| - \frac{1}{2}\delta\right), |y_i - \hat{y}_i| > \delta$$

where,

n = total number of data points

y = the actual value of the data point. Also known as true value.

$\hat{y}$ = the predicted value of the data point. This value is returned by the model.

δ = defines the point where the Huber loss function transitions from a quadratic to linear.

Remember:

- MSE is used when there are no outliers.
- MAE is used when there are outliers but the function is not differentiable.
- Huber loss is used when you have around 30% outliers in dataset.

Classification Loss

1. Binary Cross Entropy / Log Loss :

$$\text{Log loss} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)$$

Log loss works when we have binary output.
So,
When output = 0 ;

$$\text{log loss} = -\frac{1}{N}\sum_{i=1}^{N} \log(1-\hat{y}_i)$$

When output = 1 ;

$$\text{log loss} = -\frac{1}{N}\sum_{i=1}^{N} \log \hat{y}_i$$

2. Categorical Cross Entropy :

$$\text{loss} = -\sum_{j=1}^{K} y_j \log(\hat{y}_j)$$

where,
K is number of classes in the data.

- If target column has one hot encode to classes like 001, 010, 100 then use categorical cross entropy.
- If the target column has numerical encoding to classes like 1, 2, 3, 4 .... N then use sparse categorical cross-entropy.

6. Optimizer in Neural Network

- Optimizers are algorithms or methods used to change the attributes of your neural network such as weights and learning rate in order to reduce the losses.

- Used when we have lots of local minimums and a global minimum where algo can stuck on local minimum.

- Some Optimizers are:
  - Gradient descent
  - Stochastic gradient descent
  - Stochastic gradient descent with momentum
  - Mini-batch gradient descent
  - Adagrad
  - RMSProp
  - AdaDelta
  - Adam

7. Improve the Performance of Neural Network

- Hyperparameters Tuning:
  - No. of hidden layers
  - No. of neurons per layer
  - Learning rate
  - Optimizer
  - Batch size

- Activation function
- Epochs

↳ Solving Vanishing / Exploding Gradient :
- Optimizer manipulation
- Changing initialization weight
- Batch normalization.

↳ Use Big Data :
- Use transform learning to increase size of data.

↳ Overcome Slow Training :
- Can be achieved by hyperparameter tuning.
- Solve vanishing / exploding gradient.

↳ Control Overfitting :
- L1 and L2 regularization
- Early stopping
- Batch normalization.

8. Overfitting in Deep Learning

↳ Overfitting is a common explanation for the poor performance of a predictive model.

↳ Overfitting refers to an unwanted behaviour of a machine learning algorithm used for predictive modeling.

Minimization of Overfitting

- Cross validation
- Train with more data
- Remove features
- Early stopping
- Regularization (L1 and L2)
- Ensembling
- Hyperparameter tuning

Early Stopping

- Early stopping is a regularization technique in machine learning to prevent overfitting.
- It involves monitoring the model's performance on a validation set during training and stopping the training process when the performance on the validation set starts to degrade.

Regularization

- Regularization is a technique used in machine learning to prevent overfitting by adding a penalty to the loss function.
- This penalty discourage the model from becoming too complex and helps it generalize better to new, unseen dataset.

Date
Page 222

Working Practically

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

dataset = pd.read_csv("ChurnModeling.csv")
dataset.head(3)

# Drop columns with object (string) type data
dataset = dataset.select_dtypes(exclude=["object"])
dataset.head(3)

# Check for null values
dataset.isnull().sum()  # No null values

# Seperate input and output data
input_data = dataset.iloc[:, :-1]
output_data = dataset.iloc[:, -1]

# Scaling the input data
from sklearn.preprocessing import StandardScalar

ss = StandardScalar()
input_data = pd.DataFrame(ss.fit_transform(input_data),
                          columns = input_data.columns)

# Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(input_data, output_data, test_size = 0.2, random_state = 12)
```

```
# Create an artificial neural network
import tensorflow
from keras.layers import Dense
from keras.models import Sequential

ann = Sequential() # Object initialization

# Check the shape of training data
print(x_train.shape, y_train.shape) # Ensure shapes are
# Correct

# Build different layers
ann.add(Dense(8, input_dim = x_train.shape[1],
              activation = "relu")) # first hidden layer
# First parameter 8 is the number of nodes in the
# layer
ann.add(Dense(6, activation = "relu") # Second hidden
# layer
ann.add(Dense(4, activation = "relu") # Third hidden
#layer
ann.add(Dense(2, activation = "relu") # Fourth hidden
# layer.
ann.add(Dense(1, activation = "sigmoid")) # Output
#layer

# Compile the model
ann.compile(optimizer = "adam", loss = "binary_crossentropy"
            , metrics = ["accuracy"])

# Train the model
ann.fit(x_train, y_train, batch_size = 100, epochs = 50,
```

validation_data = (x_test, y_test)
# Validation data is for testing accuracy.

View training history
ann.history.history
The history attribute of a keras model contain
the training history, including metrics like loss and
accuracy for each epoch.

# To get all the keys of history
ann.history.history.keys()

To see specific values of keys
ann.history.history["accuracy"] # For accuracy

# To get training and testing accuracy of each epoch
train_accuracy = ann.history.history["accuracy"]
test_accuracy = ann.history.history["val_accuracy"]

# Visualize the accuracy
# Get the lengths of accuracies for plotting
len(test_accuracy), len(test_accuracy)

Draw graph
plt.plot([i+1 for i in range(len(test_accuracy))],
test_accuracy, color="blue")
plt.plot([i+1 for i in range(len(train_accuracy))],
train_accuracy, color="red")
plt.show()

Check the accuracy of our model

```
from sklearn.metrics import accuracy_score

# Predictions of x_train
predict_x_train = ann.predict(x_train)
x_train_prediction = []
for i in predict_x_train:
    if i[0] > 0.5:
        x_train_predictions.append(1)
    else:
        x_train_predictions.append(0)
# View predictions
x_train_predictions

# Predictions of x_test
predict_x_test = ann.predict(x_test)
x_test_predictions = []
for i in predict_x_test:
    if i[0] > 0.5:
        x_test_predictions.append(1)
    else:
        x_test_predictions.append(0)
# View predictions
x_test_predictions

# Get accuracy
train_data_accuracy = accuracy_score(y_train, x_train_predictions) * 100
test_data_accuracy = accuracy_score(y_test, y_test_predictions) * 100
train_data_accuracy, test_data_accuracy
```

""" If the training accuracy is more than testing accuracy, the model is overfitted.
To avoid overfitting we use various techniques like Early Stopping and Regularization.
These methods help in minimizing the gaps between the accuracies. """

Early Stopping :

```
from keras.callbacks import EarlyStopping

# Train the model
ann.fit (x_train, y_train, batch_size = 100, epochs= 50,
         validation_data= (x_test, y_test ), callbacks =
                EarlyStopping ())
# EarlyStopping stops training if epoch is going towards
# overfitting

# To get training and testing accuracy
train_accuracy = ann.history.history ["accuracy"]
test_accuracy = ann.history.history ["val_accuracy"]

# Visualize the accuracy
plt.plot ([i+1 for i in range (len (test_accuracy))],
          test_accuracy , color = "blue")
plt.plot ([i+1 for i in range (len (train_accuracy))],
          train_accuracy , color = "red")
plt.show()
```

Regularization :

```
from keras.regularizers import L2

# Build different layers
ann.add (Dense (8, input_dim = x_train.shape [1],
                activation = "relu",
                kernel_regularizer = L2 (l2 = 0.01)))
# kernel_regularizer stops the training if it is going
# towards overfitting using given regularization
# technique.
ann.add (Dense (6, activation = "relu", kernel_regularizer =
                L2 (l2 = 0.01)))
# kernel_regularizer can be used in different layers
# differently if you want or you can simply skip as
# below
ann.add (Dense (4, activation = "relu"))
ann.add (Dense (2, activation = "relu"))
ann.add (Dense (1, activation = "sigmoid"))

# Compile the model
ann.compile (optimizer = "adam", loss = "binary_
              crossentropy", metrics = ["accuracy"])
# Train the model
ann.fit (x_train, y_train, batch_size = 100, epochs = 50,
         validation_data = (x_test, y_test))

# To get training and testing accuracy
train_accuracy = ann.history.history ["accuracy"].
test_accuracy = ann.history.history ["val_accuracy"]
train_accuracy, test_accuracy
```

```
#Visualize the accuracy
plt.plot([i+1 for i in range(len(test_accuracy))],
         test_accuracy, color = "blue")
plt.plot([i+1 for i in range(len(train_accuracy))],
         train_accuracy, color = "red")
plt.show()
```

9. Batch Normalization

- Batch normalization is a technique used to improve the training of deep neuron networks by normalizing the inputs of each layer
- This helps to stabilize and accelerate the training process.

Working Practically

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

dataset = pd.read_csv("ChurnModelling.csv")
dataset.head(3)

#Drop columns with object (string) type data
dataset = dataset.select_dtypes(exclude = ["object"])
dataset.head(3)
```

```
# Check for null values
dataset.isnull().sum() # No null values

# Seperate input and output data
input_data = dataset.iloc[:, :-1]
output_data = dataset.iloc[:, -1]

# Scaling the input data
from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
input_data = pd.DataFrame(ss.fit_transform(input_data),
                          columns = input_data.columns)

# Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(
        input_data, output_data, test_size = 0.2,
        random_state = 42)

# Create an artificial neural network
import tensorflow
from keras.layers import Dense
from keras.models import Sequential
from keras.callbacks import EarlyStopping
from keras.regularizers import L2

ann = Sequential() # Object initialization

# Build different layers.
```

Date
Page 230

```
ann.add(Dense(8, input_dim=x_train.shape[1],
              activation="relu", kernel_regularizer=
              L2(l2=0.01))) # first hidden
# layer.
ann.add(Dense(6, activation="relu", kernel_regularizer
              =L2(l2=0.01)))
ann.add(Dense(4, activation="relu", kernel_regularizer=
              L2(l2=0.01)))
ann.add(Dense(2, activation="relu", kernel_regularizer=
              L2(l2=0.01)))
ann.add(Dense(1, activation="sigmoid"))

# Compile the model
ann.compile(optimizer="adam", loss="binary_crossentropy",
            metrics=["accuracy"])

# Train the model
ann.fit(x_train, y_train, batch_size=100, epochs=50,
        validation_data=(x_test, y_test), callbacks=
        EarlyStopping())

# Training and testing accuracy of each epoch
train_accuracy = ann.history.history["accuracy"]
test_accuracy = ann.history.history["val_accuracy"]
train_accuracy, test_accuracy

# Visualize the accuracy
plt.plot([i+1 for i in range(len(test_accuracy))],
         test_accuracy, color="blue")
plt.plot([i+1 for i in range(len(train_accuracy))],
         train_accuracy, color="red")
```

```
plt.show()

# Check the accuracy of our model
from sklearn.metrics import accuracy_score

# Predictions of x_train
predict_x_train = ann.predict(x_train)
x_train_predictions = []
for i in predict_x_train:
        if i[0] > 0.5:
                x_train_predictions.append(1)
        else:
                x_train_predictions.append(0)
# View predictions
x_train_predictions

# Predictions of x_test
predict_x_test = ann.predict(x_test)
x_test_predictions = []
for i in predict_x_test:
        if i[0] > 0.5:
                x_test_predictions.append(1)
        else:
                x_test_predictions.append(0)
# View predictions
x_test_predictions

# Get accuracy
train_data_accuracy = accuracy_score(y_train, x_train_
                      predictions) * 100
test_data_accuracy = accuracy_score(y_test, x_test_
```

predictions)* 100
train_data_accuracy, test_data_accuracy

""" If the training accuracy is more than testing accuracy, the model is overfitted.
To avoid overfitting we use Batch Normalization to minimize the gaps between the accuracies. """

Batch Normalization :

```
from keras.layers import BatchNormalization

# Build different layers
ann.add (Dense (8, input_dim = x_train.shape[1],
                activation = "relu",
                kernel_regularization = L2 (l2 = 0.01)))
ann.add (BatchNormalization()) # Applying batch
# normalization on first hidden layer output
ann.add (Dense (6, activation = "relu",
                kernel_regularization = L2 (l2 = 0.01)))
ann.add (BatchNormalization()) # Applying batch
# normalization on second hidden layer output.
ann.add (Dense (4, activation = "relu",
                kernel_regularization = L2 (l2 = 0.01)))
ann.add (BatchNormalization()) # Applying batch
# normalization on third hidden layer output.
ann.add (Dense (2, activation = "relu",
                kernel_regularization = L2 (l2 = 0.01)))
ann.add (BatchNormalization()) # Applying batch
# normalization on fourth hidden layer.
ann.add (Dense (1, activation = "sigmoid"))
```

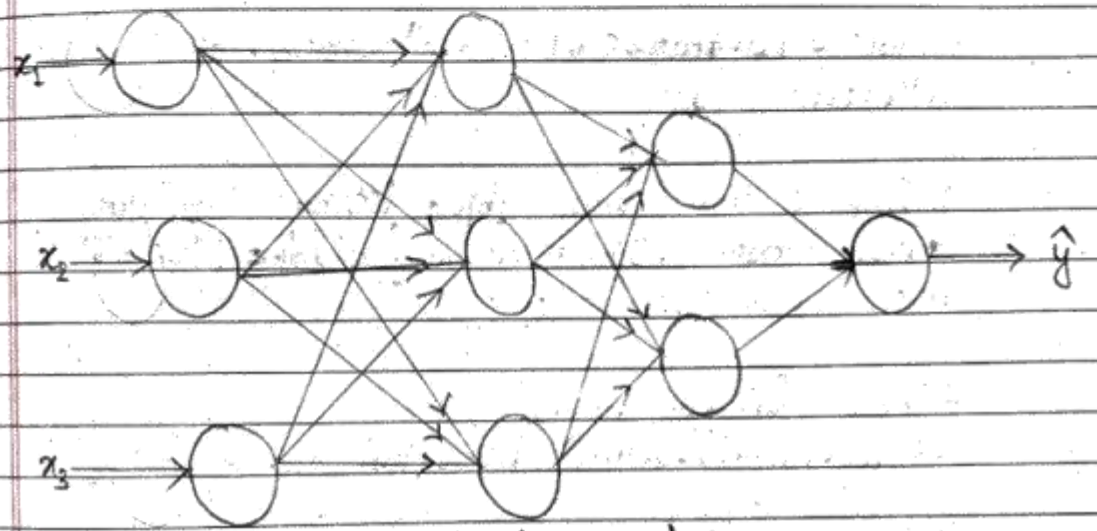
```
# Batch normalization is not required in output
# layer

# Compile the model
ann.compile(optimizer="adam", loss="binary-
            crossentropy", metrics=["accuracy"])

# Train the model
ann.fit(x_train, y_train, batch_size=100, epochs=50,
        validation_data=(x_test, y_test),
        callbacks=EarlyStopping())

# Training and testing accuracy of each epoch
train_accuracy = ann.history.history["accuracy"]
test_accuracy = ann.history.history["val_accuration"]
train_accuracy, test_accuracy

# Visualize the accuracy
plt.plot([i+1 for i in range(len(test_accuracy))],
         test_accuracy, color="blue")
plt.plot([i+1 for i in range(len(train_accuracy))],
         train_accuracy, color="red")
plt.show()
```

10. Dropout Layer

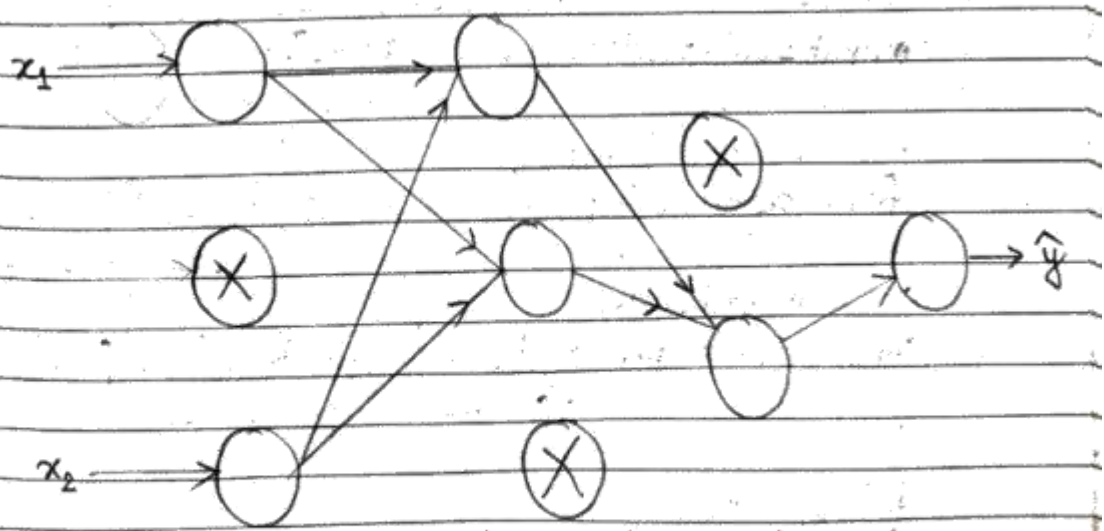
↳ All the forward and backward connections with a dropped node are temporarily removed, thus creating a new network architecture out of

the parent network.

6 The nodes are dropped by a dropout probability of p.



(a) Standard Neural Net



(b) After applying dropout layer

Working Practically

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

dataset = pd.read_csv("ChurnModeling.csv")
dataset.head(3)

#Drop columns with object (string) type data
dataset = dataset.select_dtypes(exclude=["object"])
dataset.head(3)

#Check for null values
dataset.isnull().sum() #No null values

#Seperate input and output data.
input_data = dataset.iloc[:, :-1]
output_data = dataset.iloc[:, -1]

#Scaling the input data
from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
input_data = pd.DataFrame(ss.fit_transform(input_data),
                          columns = input_data.columns)

#Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(
        input_data, output_data, test_size=0.2,
```

```
random_state = 42)

#Create an artificial neural network
import tensorflow
from keras.layers import Dense, BatchNormalization
from keras.models import Sequential
from keras.callbacks import EarlyStopping
from keras.regularizers import L2

ann = Sequential() # Object initialization

#Build different layers
ann.add(Dense(8, input_dim = x_train.shape[1],
              activation = "relu",
              kernel_regularizer = L2(l2 = 0.01)))
ann.add(BatchNormalization())
ann.add(Dense(6, activation = "relu",
              kernel_regularizer = L2(l2 = 0.01)))
ann.add(BatchNormalization())
ann.add(Dense(4, activation = "relu",
              kernel_regularizer = L2(l2 = 0.01)))
ann.add(BatchNormalization())
ann.add(Dense(2, activation = "relu",
              kernel_regularizer = L2(l2 = 0.01)))
ann.add(BatchNormalization())
ann.add(Dense(1, activation = "sigmoid"))

#Compile the model
ann.compile(optimizer = "adam", loss = "binary-
            crossentropy", metrics = ["accuracy"])
```

```
# Train the model
ann.fit (x_train, y_train, batch_size = 100, epochs=
        50, validation_data = (x_test, y_test),
        callbacks = EarlyStopping ())

# Training and testing accuracy of each epoch
train_accuracy = ann.history.history ["accuracy"]
test_accuracy = ann.history.history ["val_accuracy"]
train_accuracy, test_accuracy

# Visualize the accuracy
plt.plot ([i+1 for i in range (len(test_accuracy))],
        test_accuracy, color = "blue")
plt.plot ([i+1 for i in range (len(train_accuracy))],
        train_accuracy, color = "red")
plt.show ()

# Check the accuracy of our model
from sklearn.metrics import accuracy_score

# Predictions of x_train
predict_x_train = ann.predict (x_train)
x_train_predictions = []
for i in predict_x_train :
        if i[0] > 0.5 :
                x_train_predictions.append (1)
        else:
                x_train_predictions.append (0)
# View predictions
x_train_predictions
```

```
# Prediction of x-test
predict_x_test = ann.predict(x_test)
x_test_predictions = []
for i in predict_x_test:
    if i[0] > 0.5:
        x_test_predictions.append(1)
    else:
        x_test_predictions.append(0)
# View predictions
x_test_predictions

# Get accuracy
train_data_accuracy = accuracy_score(y_train,
                      x_train_predictions)*100
test_data_accuracy = accuracy_score(y_test,
                     x_test_predictions)*100
train_data_accuracy, test_data_accuracy

""" If the training accuracy is more than testing
accuracy then the model is overfitted.
To avoid overfitting, we can use Dropout layer to
minimize the gap between the accuracies. """
```

Dropout Layer:

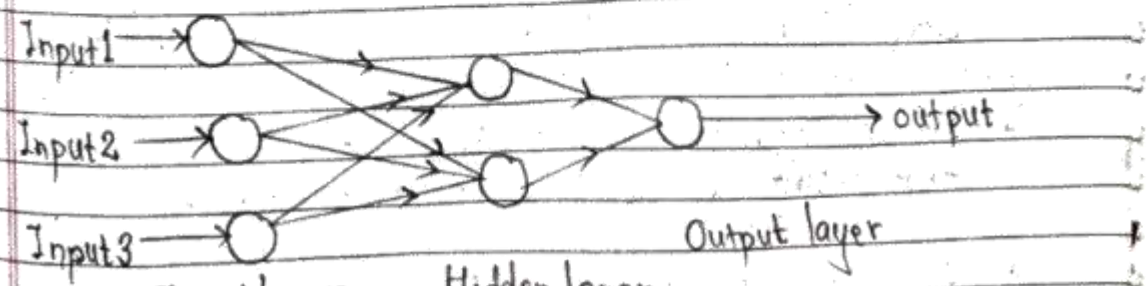
```
from keras.layers import Dropout

# Build different layers
ann.add(BatchNormalization())
ann.add(Dense(8, input_dim = x_train.shape[1],
        activation = "relu",
```

```
                kernel_regularizer = L2 (l2 = 0.01)))
ann.add (Dropout (0.3)) # Dropout layer with 30%
# to first layer
ann.add (Batch Normalization ())
ann.add (Dense (6, activation = "relu",
                kernel_regularizer = L2 (l2 = 0.01)))
ann.add (Dropout (0.3)) # Dropout layer to second
# layer
ann.add (Batch Normalization ())
ann.add (Dense (4, activation = "relu",
                kernel_regularizer = L2 (l2 = 0.01)))
ann.add (Dropout (0.3)) # Dropout layer to third
# layer
ann.add (Batch Normalization ())
ann.add (Dense (2, activation = "relu",
                kernel_regularizer = L2 (l2 = 0.01)))
ann.add (Dropout (0.3)) # Dropout layer to fourth
# layer
ann.add (Batch Normalization ())
ann.add (1, activation = "sigmoid"))

# Compile the model
ann.compile (optimizer = "adam", loss = "binary-
             crossentropy", metrics = ["accuracy"])

# Train the model
ann.fit (x_train, y_train, batch_size = 100, epochs = 50,
         validation_data = (x_test, y_test), callbacks =
         EarlyStopping ())

# Training and testing accuracy of each epoch
```

```
train_accuracy = ann.history.history["accuracy"]
test_accuracy = ann.history.history["val_accuracy"]
train_accuracy, test_accuracy

#Visualize the accuracy
plt.plot([i+1 for i in range(len(test_accuracy))],
         test_accuracy, color="blue")
plt.plot([i+1 for i in range(len(train_accuracy))],
         train_accuracy, color="red")
plt.show()
```

11. Vanishing Gradient Problem

- The vanishing gradient problem is encountered when training artificial neural networks with gradient-based learning methods and backpropagation.
- In such methods, during each iteration of training each of the neural network's weights receives an update proportional to the partial derivative of the error function with respect to the current weight.



Solutions for Vanishing Gradient Problem

1. Proper weight initialization
2. Using non-saturating activation functions (relu)
3. Batch normalization
4. Gradient clipping

Working Practically

""" While working with deep learning algorithms if you are facing a problem in which the loss from each or many number of epohs comes constant.
This is vanishing gradient problem as no an gradient is seen.
The above given methods can be used in such time to overcome the problem. """

12. Hyperparameter Tuning

- Hyperparameter tuning is a crucial step in optimizing deep learning models.
- It involves finding the best set of hyperparameters that result in the most effictive model.
- It is used to increase accuracy.

Key Hyperparameters

1. **Learning Rate:** Controls how much to change the model in response to the estimated error each time the model weights are updated.

2. **Batch Size:** Number of training examples utilized in one iteration.

3. **Number of Epochs:** Number of complete passes through the training dataset.

4. **Number of Layers and Neurons:** Structure of the neural network, including the number of layers and the number of neurons in each layer.

5. **Activation Function:** Function that introduce non-linearity to the model.

6. **Optimizer:** Algorithm used to update the weights of network (e.g. Adam, SGD)

7. **Dropout Rate:** Fraction of input units to drop during training to prevent overfitting.

8. **Regularization Parameters:** Parameters like L2, L1. L1 regularization to prevent overfitting.

Methods for Hyperparameter Tuning

1. Grid Search: Exhaustively searches through a manually specified subset of the hyperparameter space.

2. Random Search: Randomly samples the hyperparameter space and performs better than grid search in practice.

3. Bayesian Optimization: Uses a probabilistic model to find the best hyperparameters.

4. Hyperband: Combines random search with early stopping to efficiently find the best hyperparameters.

5. Genetic Algorithm: Uses principles of natural selection to search for optimal hyperparameters

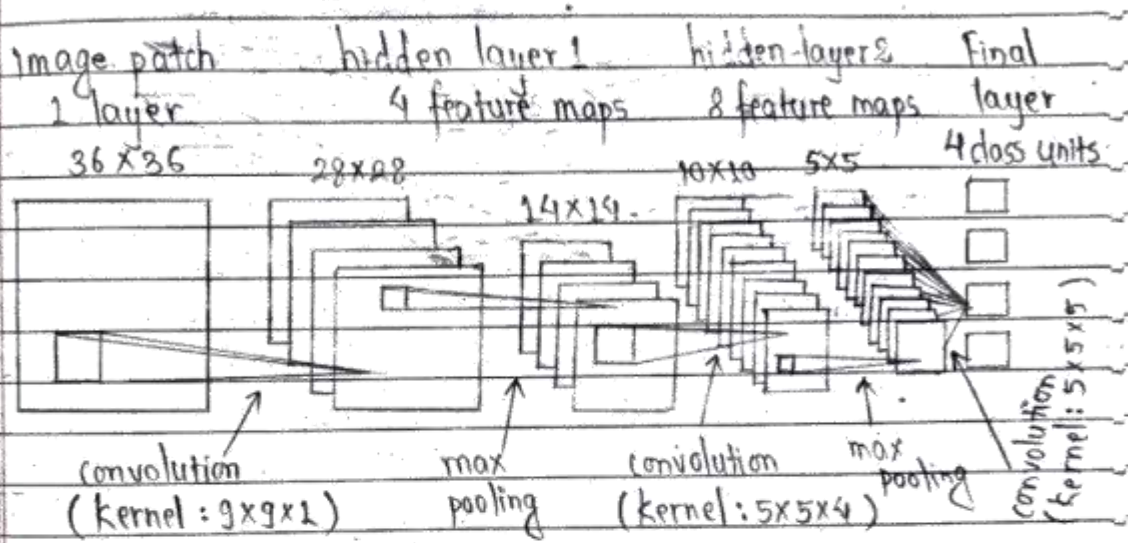
13. Convolutional Neural Network (CNN)

↳ CNN is a type of artificial neural network, which is widely used for image/object recognition and classification.

↳ Deep learning thus recognizes objects in an image by using a CNN.

↳ This works like our eyes for capturing data and brain for predictions.

Convolution → Max Pooling → Flattening → Fully Connected layers



1. Convolution

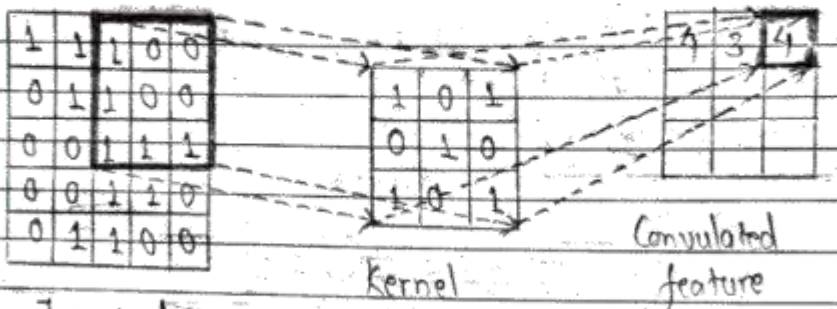
- Now in mathematics convolution is a mathematical operation on two functions that produces a third function that expresses how the shape of one is modified by the other.
- Mathematically,

$$c = (f * g)(t) = \int_0^t f(\tau)\, g(t-\tau)\, d\tau$$

Where,

c = convolution output function
f = original output function
g = function that is shifted over the input

function.
t = variable representing range of shifting
τ = shifting against t



Input data

Kernel

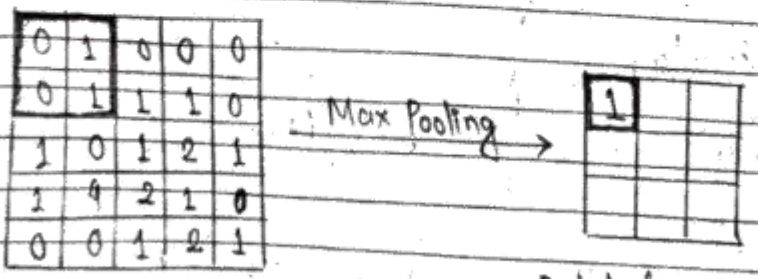
Convulated feature

1*1 = 1
0*0 = 0
0*1 = 0
1*0 = 0
1*1 = 1
0*0 = 0
1*1 = 1
1*0 = 0
+ 1*1 = 1
4

2 Pooling

10

- A pooling layer is another building block of a CNN.
- It's function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network.



Feature map

Pooled feature map

max (0, 1, 0, 1)
= 1

3. Flattening

- Flattening is converting the data into a 1-dimensional array for inputing it to next layer.
- Flatten output of the convolutional layers to create a single long feature vector.
- It is connected to the final classification model, which is called a fully connected layer.

1	1	0
4	2	1
0	2	1

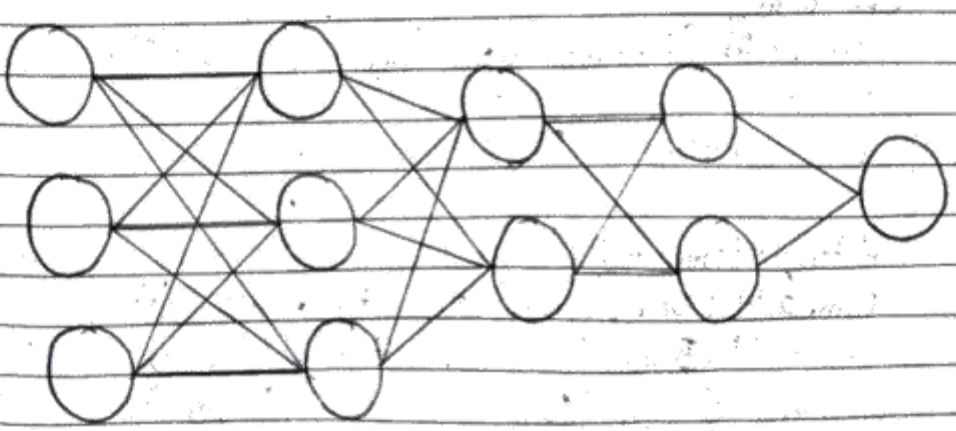
→ flattening →

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	y
1	1	0	4	2	1	0	2	1	-

pooled feature

4. Fully Connected Layers

- It is an Artificial Neural Network (ANN)



Working Practically

```
import tensorflow
from keras.layers import Dense, Conv2D, MaxPooling2D,
                         Flatten
from keras.models import Sequential
from tensorflow.keras.preprocessing.image import
                         ImageDataGenerator

#Create a CNN network
cnn = Sequential()

#Convolution
cnn.add(Conv2D(filters=32, kernel_size=(3,3),
               input_shape=(64, 64, 3),
               activation="relu"))
#Pooling
cnn.add(MaxPooling2D(pool_size=(2,2)))
#Convolution
cnn.add(Conv2D(filters=32, kernel_size=(3,3),
               activation="relu"))
#Pooling
cnn.add(MaxPooling2D(pool_size=(2,2)))
#Flatten
cnn.add(Flatten())

#Fully Connected Layers
cnn.add(Dense(64, activation="relu"))
cnn.add(Dense(32, activation="relu"))
cnn.add(Dense(16, activation="relu"))
cnn.add(Dense(8, activation="relu"))
```

```
cnn.add(Dense(4, activation="relu"))
cnn.add(Dense(2, activation="relu"))
cnn.add(Dense(1, activation="sigmoid"))

# Compile model
cnn.compile(optimizer="adam", loss="binary_crossentropy")

# Create an instance of ImageDataGenerator
train_datagen = ImageDataGenerator(
        rescale=1.0/255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True
)
# Same for test_datagen
test_datagen = ImageDataGenerator(rescale=1.0/255)

# Generate batches of augmented image data from
#the specified directory.
train_generator = train_datagen.flow_from_directory(
        r".\Pictures\Train",
        target_size=(64,64),
        batch_size=32,
        class_mode="binary"
)
# Same for test_generator
test_generator = test_datagen.flow_from_directory(
        r".\Pictures\Test",
        target_size=(64,64),
        batch_size=32,
        class_mode="binary"
```

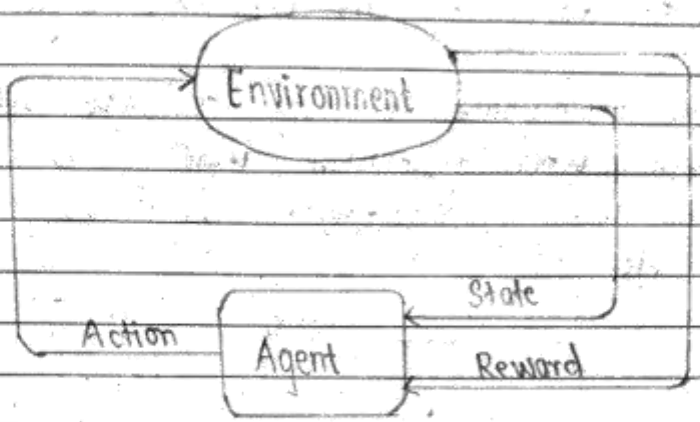
```
)

# Train the model using fit method
cnn.fit(
    train_generator,
    steps_per_epoch = 20,
    epochs = 5,
    validation_data = test_generator
)

# Evaluate the model on the test data
results = cnn.evaluate(test_generator)
results

# Making predictions (new)
from keras.preprocessing import image

# Load image
img = image.load_img(r".\Pictures\Predict\Dogs\
    PredictDog1.jpg", target_size = (64, 64))
img

# Convert the image to array
img = image.img_to_array(img)
img

# Change the dimensions of image
import numpy as np

img = np.expand_dims(img, axis=0)
```

```
cnn.predict(img)  # New prediction

# To get the output in desired form
p = cnn.predict(img)
if p[0][0] < 0.5:
        print("Dog")
else:
        print("Cat")

""" Predict the multiple values from the folder """

# Create an ImageGenerator for prediction
predict_datagen = ImageGenerator(rescale = 1.0/225)

# Generate batches of images from the specified
# directory for prediction
predict_generator = predict_datagen.flow_from_directory(
        r"\Pictures\Predict",
        target_size = (64, 64), batch_size = 32,
        class_mode = None,
        shuffle = False
)

# Make predictions
predictions = cnn.predict(predict_generator)

# Convert predictions to class labels
threshold = 0.5
predicted_classes = np.where(predictions > threshold,
                             "Dog", "Cat")
```

```
# Get the filenames
filenames = predict_generator.filenames

# Combine files and predictions
results = list(zip(filenames, predicted_classes))

# Print the results
for filename, prediction in results:
    print(f"File: {filename}, Predicted Class:
          {prediction}")
```

10. Reinforcement Learning

1. Introduction



- Reinforcement learning is a feedback-base machine learning technique where an agent learns to which actions to perform by looking at the environment and the results of actions.

- For each correct action, the agent gets positive feedback, and for each incorrect action, the agent gets negative feedback or penalty.

- The agent interacts with the environment and identifies the possible actions he can perform.

- The primary goal of an agent in reinforcement learning is to perform actions by looking at the environment and get the maximum positive rewards.

- In Reinforcement learning, the agent learns automatically using feedbacks without any labeled data, unlike supervised learning.

- Since there is no labeled data, so the agent is bound to learn by its experience only.
- Reinforcement learning is used to solve specific type of problem where decision making is sequential, and the goal is long-term, such as game playing, robotics, etc.

2. Types of Reinforcement Learning

There are two types of reinforcement learning:

1. Positive reinforcement learning
2. Negative reinforcement learning

1. Positive Reinforcement Learning

- Positive reinforcement involves adding a rewarding stimulus after a desired behaviour is performed, which increases the likelihood of behaviour being repeated.
- It is recurrence of behaviour due to positive rewards.
- Rewards increase strength and frequency of a specific behaviour.

2. Negative Reinforcement Learning:

- Negative reinforcement involves removing an aversive stimulus after a desired behaviour is performed, which also increases the likelihood of that behaviour being repeated.
- In negative reinforcement learning, negative rewards are used as a deterrent to weaken the behaviour and to avoid it.
- Reward decreases strength and the frequency of a specific behaviour.

8. Use Cases of Reinforcement Learning

Reinforcement learning (RL) is a powerful technique used in various real-world applications where decision making is crucial.

Here are some areas where RL is commonly applied:

1. Robotics

RL is extensively used in robotics for tasks such as:

- Industrial Automation: Robots learn to perform complex tasks like assembly, welding and painting by optimizing their actions to maximize efficiency and

accuracy.

- Autonomous Navigation: Robots learn to navigate through environments, avoiding obstacles and reaching targets efficiently.

2. Autonomous Vehicles:

Self driving cars use RL to make real-time decisions, such as:

- Path Planning: Determining the optimal route to a destination while avoiding obstacles and following traffic rules.
- Control Systems: Managing acceleration, braking, and steering to ensure safe and efficient driving.

3. Games Playing

RL has successfully applied to various games, including

- Video Games: Agents learn to play games like chess, go and atari games at superhuman levels by optimizing their strategies through trial and error.
- Board Games: RL algorithms have been used to develop AI that can compete and defeat human champions.

4. Health Care

In health care RL is used for:

- Personalized Treatment Plans: Developing customized treatment strategies for patients based on their unique medical histories and responses to treatments.
- Robotics Surgery: Assisting surgeons by optimizing the movements of surgical robots to improve precision and outcomes.

5. Finance

RL is applied in finance for:

- Algorithmic Trading: Developing trading strategies that maximize returns by learning from market data and trends.
- Portfolio Management: Optimizing the allocation of assets in a portfolio to balance risk and return.

6. Natural Language Processing. (NLP)

In NLP, RL is used for:

- Dialogue Systems: Training chatbots and virtual assistants to provide more accurate and contextually relevent responses.

- **Machine Translation:** Improving the accuracy of translating text from one language to another by learning from feedback.

7. Education

RL is used to create adaptive learning systems such as:

- **Personalized Learning:** Tailor educational content and pacing to individual students' needs and learning styles.
- **Intelligent Tutoring Systems:** Provide customized feedback and support to students based on their performance.

4. Q-Table

- A Q-table is a matrix used in reinforcement learning to store the expected future rewards for each state-action pair.
- Each row represents a state and each column represents an action.
- The values in the table indicates the quality (Q-value) of taking a particular action in particular state.

Structure of a Q-Table.

State	Action: Up	Action: Down	Action: Left	Action: Right
0	0.5	0.2	0.1	0.4
1	0.3	0.6	0.2	0.5
2	0.4	0.3	0.7	0.2
3	0.6	0.1	0.3	0.8

- **Rows:** Represents different states of the environment.
- **Columns:** Represents the possible action the agent can take.
- **Values:** Represent the expected future rewards (Q-values) for taking a specific action in a specific state.

Updation in Q-table

- The Q-values are updated using Q-learning algorithm.
- Q-learning is a model-free, value-based reinforcement learning algorithm used to find the optimal action-selection policy for any given finite Markov decision process (MDP).
- Q-learning helps an agent learn to maximize the total reward over time through repeated interactions

with the environment, even when the model of that environment is not known.

- Mathematically, the Q-value is updated using Bellman equation also called Q-learning formula:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left| r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right|$$

where,

- $Q(s,a)$ is the current Q-value for state (s) and action (a).
- α is the learning rate, which determines how much new information overrides the old information.
- r is the reward received after taking action (a).
- γ is the discount factor, which determines the importance of future rewards.
- $\max_{a'} Q(s',a')$ is the maximum value of Q-value for the next state (s')

- Steps in Q-learning:

1. Initialize the Q-table with zeros.
2. Observe the current state (s).
3. Choose an action (a) using an exploration - exploration strategy (e.g. epsilon-greedy)
4. Perform the action (a) and observe the reward (r) and next state (s').
5. Update the Q-value using the Q-learning formula.
6. Repeat the process until the Q-table converges or

for a fixed number of episodes.

Working Practically

```
import numpy as np

# Define the grid world
grid = [
    ['S', 'O', 'O', 'O', 'G'],
    ['O', 'X', 'O', 'X', 'O'],
    ['O', 'X', 'O', 'X', 'O'],
    ['O', 'O', 'O', 'X', 'O'],
    ['O', 'X', 'O', 'O', 'O']
]

# Define the state and action space
num_states = 25    # 5x5 grid
num_actions = 4    # Up, Down, Left, Right

# Initialize Q-table with zeros
Q = np.zeros((num_states, num_actions))

# Parameters
alpha = 0.8    # Learning rate
gamma = 0.95   # Discount factor
epsilon = 0.1  # Exploration rate

# Define the reward function
def get_reward(state):
    row, col = divmod(state, 5)
```

```
    if grid[row][col] == 'G':
        return 1
    elif grid[row][col] == 'X':
        return -1
    else:
        return 0

# Define the next state function
def get_next_state(state, action):
    row, col = divmod(state, 5)
    if action == 0 and row > 0:  # Up
        row -= 1
    elif action == 1 and row < 4:  # Down
        row += 1
    elif action == 2 and col > 0:  # Left
        col -= 1
    elif action == 3 and col < 4:  # Right
        col += 1
    return row * 5 + col

# Training the agent
for episode in range(1000):
    state = 0  # Start position
    done = False

    while not done:
        if np.random.uniform(0, 1) < epsilon:
            action = np.random.choice(num_actions)
            # Explore
        else:
            action = np.argmax(Q[state, :])
```

#Exploit

```
next_state = get_next_state(state, action)
reward = get_reward(next_state)
Q[state, action] = Q[state, action] + alpha *
                   (reward + gamma * np.max(
                   Q[next_state, :]) -
                   Q[state, action])

state = next_state

if grid[divmod(state, 5)[0]][divmod(
        state, 5)[1]] == 'G' or grid[
        divmod(state, 5)[0]][divmod(
        state, 5)[1]] == 'x':
    done = True

print("Trained Q-table:")
print(Q)
```

Date
Page

Congratulations!

You've completed Basic Data Science!

Thank you for reaching the end.
Do share with friends if you find it helpful.
Share your feedback if possible.

Get more information about Basic Data Science :
https://github.com/KushalPrasadJoshi/basic-data-science

Get more study materials here:
https://github.com/KushalPrasadJoshi/study-resources

You can follow me on LinkedIn here:
https://www.linkedin.com/in/kushal-prasad-joshi/