Data Science

**The approach for statistical analysis.
**Data science is all about extracting insights from data using various techniques like statistics, machine learning, and data visualization.

AI

*refers to the development of intelligent systems that can learn, adapt, and make decisions by analyzing large volumes of structured and unstructured data.

*This integration of AI and Data Science enables machines to extract patterns, recognize trends, and generate insights without explicit human programming.

ML

*Machine Learning (ML) is a subset of Artificial Intelligence (AI) that enables computers to learn from data and make predictions or decisions without being explicitly programmed. [No **hardcoded solutions**]

* ML models use statistical techniques to identify patterns and improve performance over time.

Supervised Learning:

*The data trained here is labeled (input-output pairs).

***Labeled data** is a dataset where each data point has both **an input** (features) and a **corresponding output** (label/target value).

*This is used to classify or to predict. Map inputs to correct outputs using historical examples.

UnSupervised Learning:

This data is unlabeled and it is used to make a group or find a pattern.

*Only input is here. No specific output.

*Then we will make groups by finding the pattern [**clusterring**].

***Unlabeled data** refers to data that does **not have predefined categories or labels**. Consists of raw input data without corresponding target values (outputs).

Neural Network

*Neural Networks simulate the human brain and allow machines to learn complex patterns. They are the backbone of Deep Learning,

enabling cutting-edge AI applications.

*When **multiple nodes** are **interlinked** then a **net-like structure** in layers is created which we call the neural net.

*It consists of layers of artificial neurons (nodes) that process and transform input data to make predictions.

Deep Learning

Deep Learning is a subfield of Machine Learning (ML) that uses artificial neural networks with multiple layers (hence "deep") to process data and learn patterns.

*It is the technology behind AI breakthroughs like ChatGPT, self-driving cars, facial recognition, and medical diagnosis.

* Capable to focus on right

BASIC IDEAS:

In ML, weight & biases are the **fundamental parameters** of a model that help make predictions. They are **primarily used in linear models** & **neural network**.

WEIGHT

Weights **determine** the **importance** of each **input feature** in **influencing** the **prediction**.

* They scale the **input features**.
* **IT IS BASICALLY THE SLOPE OF THE EQUATION.**
* **A higher weight means the features** have **stronger impact on prediction**.
* Helps model pick up **more relevant features**.

BIAS

Bias is an **offset** that **allows the model to better fit the data** by **shifting the prediction up or down.**

* It ensures that the model **does not have go through the origin** if the **data doesn't**.
* It is the **intercept of y-axis**
* Provides **flexibility,** improving the model's fit data.

NOISE

Noise refers to **mislabeled data points** or **outliers** that **don't follow the true class distribution**.

* **errors or irregularities in data can mislead a learning algorithm.**
* Distorts the geometric relationships between points, affecting kernel computations.
* Data points far outside the true distribution (e.g., a salary entry of $1 million in a dataset of average incomes).

GENERATIVE AI:
Updated AI which can take decision and make future data from previous data.

STATISTICS

For a system two criterias need to be fulfilled.

In a machine Learning model,
Feature is the Input
Label is the output

For example:
Some criterias need to be fulfilled to hire someone.Example:
i) CV
ii) Projects
iii) CP (Comp)
iv) ECA

Then casing on the criterias 2 outcomes are expected:
I) Either getting hired (1)
II) No getting hired (0)

So the criterias here are Features and the Outcomes expected are the Labels

Linear Regression: [when only one dependent variable is available]

We take a best fit line at first. [A probable line with the best qualities]

****best fit line:** The best fit line is the straight line that minimizes the error between predicted (Y) and actual (X) values in the dataset**

1) We will first plot the given points, Y as the dependent variable and x as the independent variable.

2) Then we will find the average of all the given data (y) and plot the next unknown point there.

3) Then we will find the deviation from those points to the line that is there either negative or positive.

4) Then we can find the SSE (sum of square of error).

5) We will actually need to plot the points with the independent variables with respect to which the dependent variable fluctuates.

6) The more dependent variables we can get, the more sufficient the model. For example:
area to price graph.

7) Then, plotting them in the increasing measure we can decrease the amount of SSE which can be found after drawing the average line.

8) With minimalistic SSE, we can predict the middle values or even the future values.

How to **Linearly Regress?**

i) **Import libraries**

matplotlib.pyplot to **plot** the points.
numpy to **make arrays** of the data.
sklearn for **training** the data where,
dataset helps us **load** the data
sklearn.**metrics** for the **MSSE** [for the **mean_squared_error**]

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error
```

ii) **Load the data**

*the csv file will be loaded and stored it in a variable using:

var_name = **datasets**.load_**file_name**()

diabetes = datasets.load_diabetes()

np.newaxis needs to be used. Then, we get to **increase the dimensions** of an array which is essential for **reshaping data** in Numpy and machine learning.

Many ML libraries like scikit-learn expect 2D input, even if there is only one feature.

EX1: **1D -------> 2D column vector**

```
arr = np.array([1, 2, 3])    # Shape: (3,)
arr_column = arr[:, np.newaxis]    # Shape: (3,1)
print(arr_column)
```

EX2: **1D -------> 2D row vector**

```
arr_row = arr[np.newaxis, :]    # Shape: (1,3)
print(arr_row)
```

Our example:

*Here " : " selects all the rows and keeps all 442 samples.
* and selecting 2 gets us 2 number column from the dataset.
* Then, np.newaxis reshapes it into (442,1)

```
diabetes_X= diabetes.data[:,np.newaxis,2]
```

iii) Storing [training & testing data]:

1) We will need to save the data which need to be trained to a variable for each x and axis. This takes all the values except the last 30.

```
diabetes_X_train = diabetes_X[:-30]
diabetes_X_test = diabetes_X[-30:]
```

2) Then, we will need to do the same with y-axis. But, here we will need to take the target value which is our y value. This takes the last 30 values only.

```
diabetes_y_train = diabetes.target[:-30]
diabetes_y_test = diabetes.target[-30:]
```

For training we took 412-30 elements from the first and for training we took 30 elements using the slicing method.

iv) TRAINING DATA:

1) Storing the model:

i) We will use the linear model to call on to the Linear Regression model

ii) Then we will need to store it.

2) Fitting the values:

We will need to fit the x-axis and y-axis value within the fit function of the model after saving the model in the variable.

v) STORE PREDICTED VALUES:

1) We will need to put in the testing values within the predict function

2) Then, we will have to save the predicted values within a variable.

EX:

```
diabetes_y_predicted = model.predict(diabetes_X_test)
```

vi) OBTAINING WEIGHT & INTERCEPT:

1) We will need to call the object model name and then call for the coefficient function to get the weight using **model_var_name.coef_**

EX: `print("Weights:", model.coef_)`

2) We will need to call for the object model name and then call for the intercept function to get the intercept using **model_var_name.intercept_**

EX: `print("Intercept:", model.intercept_)`

SCATTER PLOT [DATASET VALUES]
&
LINE PLOT [EXPECTED VALUE]:

1) The scatter plot will give us the plotting of the dataset info.

2) The line plot will give us the line plotting of the Expected Line of data.

Ex:

```
plt.scatter(diabetes_X_test, diabetes_y_test)
```

```
plt.plot(diabetes_X_test, diabetes_y_predicted)
plt.show()
```

OVERALL CODE:

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error

diabetes = datasets.load_diabetes()
diabetes_X = diabetes.data[:, 2].reshape(-1, 1)

from sklearn.metrics import mean_squared_error
diabetes_X_train = diabetes_X[:-30]
diabetes_X_test = diabetes_X[-30:]

diabetes_y_train = diabetes.target[:-30]
diabetes_y_test = diabetes.target[-30:]

model = linear_model.LinearRegression()
model.fit(diabetes_X_train, diabetes_y_train)

diabetes_y_predicted = model.predict(diabetes_X_test)
print(f"Mean squared error is: {mean_squared_error(diabetes_y_test,diabetes_y_predicted)}")

print("Weights:", model.coef_)
print("Intercept:", model.intercept_)

plt.scatter(diabetes_X_test, diabetes_y_test)
plt.plot(diabetes_X_test, diabetes_y_predicted)
plt.show()
```

Linear REGRESSION [types]

There are 3 types of Linear Regression:

i) Simple Linear Regression [SLR]
ii) Multiple linear Regression [MLR]
iii) Polynomial Regression

Simple Linear Regression

When for continious output we have only one feature (1 independent variable) then is called SLR.

For Example:

CGPA----------> YEAR
3.7---------->1st
2.3---------->2nd
3.8---------->3rd

In basic linear regression, we use the normal straight line equation:
y=mx+c
where m is the slope. [With every change in the value of x how much does y change]

****INTERFERENCE****

The above line equation is function that can relate x and y for given value of x, we can find corresponding y.

Multiple linear regression

If we have two or more than **two independent variable** It is called ** EX: 3D maybe

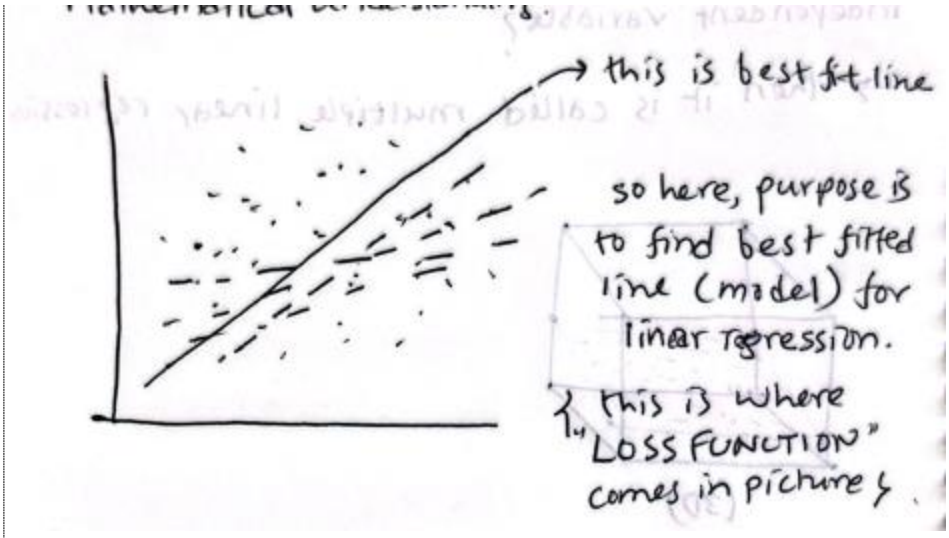
Multiple linear regression:

y = w1x1 + w2x2 + w3x3.....wnxn + b

Advantage:
i) Simple to perform
ii) Performs well on data with linear relationship

Disadvantage:
i) Not suitable for data having non-linear relationship.
ii) Underfitting issue
iii) Sensitive to outliers

Mathematical Understanding:



The purpose is to find the best fitted line (model) for linear regression.
This is where loss function comes in the picture.

"Loss Function" measures how far a **predicted value (y1)** from its **actual value (yi)**

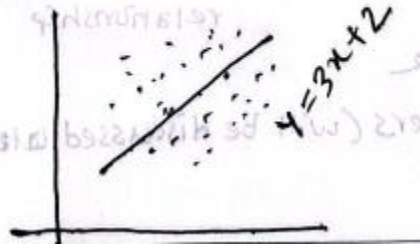
It is helpful to determine which model performs better and which parameter are better.

MSSE [Mean of sum of square of error]

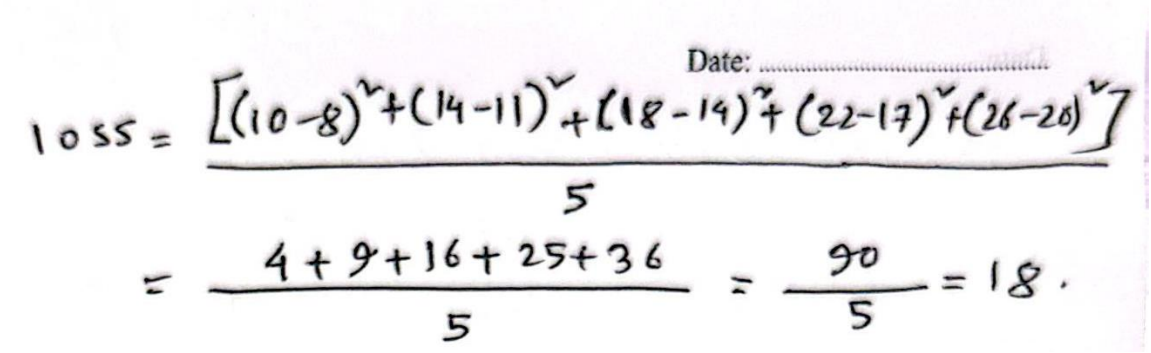
$$\text{loss} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2$$

Let try to understand with the example

$m = 3,\ c = 2$



x	y	$\hat{y}$
2	10	8
3	14	11
4	18	14
5	22	17
6	26	20



[We take the squre so that positive and negative value may not cancel each other]
low loss value--------> Higher Accuracy
high loss value--------> Low Accuracy

Optimization:

We can improve a model by optimization, A technique called "Gradient Descent", where we repeat the process iteratively until we get "minimum loss function".

Train Test Splitting:

In Machine Learning (ML), the train-test split is a technique used to evaluate the performance of a model by dividing the dataset into two (or sometimes three) parts:

1) Training Set: Used to train model
2) Testing Set: Used to evaluate the model's performance
3) Validation Set(Optional): Used to fine-tune hyperparameters before final testing.

Why should we split?

1) Overfitting: If you train and test on the same data, the model may memorize instead of learning patterns (overfitting).
2)A separate test set helps to estimate how well the model generalizes to new, unseen data.

Splits are usually made from **80(training)-20(testing)%**

Multiple Variable Regression:

Multivariable regression (also called multiple linear regression) is a type of regression analysis that **models the relationship between one dependent variable and multiple independent variables.**

When more variables intoduced the accuracy of that specific model increases more and more. But it also

means more Time complexity and more calculations.

Let us say:

Y here is a Dependent / target variable. And X can be the independent predictor variables and as the line equation is W_0 is the intercept. The rest Ws are Weights

For 1 independent variable eqn will be: $Y = W_0 + W_1X_1$
For 2 independent variables eqn will be: $Y = W_0 + W_1X_1 + W_2X_2$
For 2 independent variables eqn will be: $Y = W_0 + W_1X_1 + W_2X_2 + W_3X_3$
For n independent variables eqn will be: $Y = W_0 + W_1X_1 + W_2X_2 + W_3X_3 \ldots\ldots\ldots + W_nX_n$

Classification:

*Classification refers to the task of predicting the category or class label of an input based on its features. Its about assigning each input to one of the several predefined categories or classes.

*It is a **supervised learning problem** as the problem uses **labeled data** for training its model.

*There might be **two types** of **classification:**

i) Binary: This type of categorization is only between a yes and a no (0 & 1)

ii) **Multiclass:** The type of categorization where there might be more than 2 classes

da by definition:

* The task is to classify the input into one of more than two classes.
* Here, in a supervised system (data will be labeled i.e range of ouputs are already given)
* It is a **non-parametric** and **lazy learning** algorithm.
*It is a **simple, non-parametric machine learning algorithm** used for **classification** and **regression** tasks.
*It works by **finding the "K" closest data points** (neighbors) to a given data point, and
*then **making predictions based** on the majority class (for classification) or the average (for regression) of those neighbors.

non-parametric: It does not make any assumptions about the underlying data distribution.

Lazy Learning: *It **doesn't actually learn** a model **during the training phase**.
*It **memorizes** the **training dataset** and performs **computation** only when **given new data** for prediction.

How it works?

i)Takes a value K for making prediction.
ii) Finds the closest near values.
iii)**Makes Table:** Finds the charateristics of the values found around.
iv) **Sorts Table:** compares the appearance of those characteristics with each other.
v) **Takes the most appeared answer:** The most appeared character among the values is taken to be the prediction for the characteristic of K predicted value.

Drawbacks:

i) **Testing Burden**: As all we do is done real time in the email case scenario.
ii) **Estimation of K** is difficult
iii) **Overall increase of complexity** for **Increase of variables**

KNN Classifier by code:

Info on the dataset : Output can be 0,1,2 as 3 outputs are possible for 3 different flowers

```
#Loading required modules

from sklearn import datasets                                    #importing dataset
from sklearn.neighbors import KNeighborsClassifier      #importing model

iris= datasets.load_iris()      #loading dataset

#print(iris.DESCR)                  #printing description of dataset

features = iris.data
#we call for data subsection to get features [2D array of data]

labels = iris.target
#we call for the ouputs returned in 1D array [predetermined]

clf = KNeighborsClassifier()                # 1) loading model
clf.fit(features, labels)                   # 2) Fitting model
preds = clf.predict([[3.1,1,1,1]])      # 3) predicting output using model
```

```
print(preds)
```

 # Right now the outputs are given according to our query, then when we give a different value within the predict part, we will get a different output, as the values change for input the output also changes as said before "It computes only when new data is introduced"

OVERFITTING
&
KNN

Overfitting :

- Overfitting is when, a model learns the training data too well, including noise and random fluctuations, making it perform poorly on new data.
- Higher accuracy with Training data but low accuracy with validation / test data.
- Model captures unnecessary details and fails to **generalize**.

Basic Understanding & why should we avoid it:

** IF the model learns the data bit too well that it follows it to the T, it **becomes a bit too stiff to the original data**. So, when we bring in **too much data within consideration** it only acts **like those data patterns** and then if new data is introduced for prediction value it only follows the stiff old trend from before.**

[**overfitting**]
[**Low training error**]
[**high testing error**]
[**SSE tends to 0**]

FIX:
While Training:

The better we learn the pattern of the appearance of the data copying its movement is our aim but not following its footsteps directly but learn to learn the trends and patterns themselves.

Underfitting:

*when a model is too simple to capture the underlying patterns in the data. It performs poorly on both training and test data.

*__Low accuracy__ on both __training__ and __test__ data.

*Too __simple model__

LOGISTIC REGRESSION

* It returns PROBABILITY.

* Logistic regression is a statistical method used for **binary classification** (Yes or No) problems, where the **outcome variable** is **categorical** and consists of two classes.

*It is used to **model the relationship** between **a dependent binary variable** and **one or more independent variables**.

Outcome Explanation [**When to use linear regression**]

*It returns how much probability is there for an occurence to happen. [0/1]

1) [0 shows what we call is Impossible]

2) [within 0 and 1 we get whats most likely whats most unlikely bleh bleh]

3) [1 shows what we call is the universal truth]

4) But here it crosses the limit from 0 to 1 for which **logistic Regression comes in rolling.**

Generalized equation of Logistic Regression:

In logistic regression, the basic formula (linear regression output) is transformed into a probability using the sigmoid function.

Basic Formula:

$$z = W^T x + b$$

Here,

w= weight of each feature [importance]
x = **input features**
b = Bias term [shifter of decision boundaries]

** Here, we always take input as a **numpy array** , we transpose these arrays and then get W of t and at the same time b is also a numpy array**

For example:

* x = [x1,x2] [Input Features]
* weight = [w1, w2] [Weights]

Expected probability (Label) = alpha + beta*x1 + gamma*x2

Here, x1 and x2 are the features or **independent variables** and amongst them the beta and gamma are weights.

HOW ALGORITH ON THE BASIS OF THIS WORKS:

* Here, the model first learns these parameters, that is the **weight and the biases** .
* Then, it **predicts.**
* The value we get from y, is called the **log of odds** .
* we put it in the **sigmoid function**.

Log of ODDS:

* IF the probabilty of anything happening or not is p.
* Then we will find the odds with the help of:

$$ODDS= p/(1-p)$$

* Then if we log the resultant odd we will get **log of odds**.

    * Here, **p is the probabilty from the sigmoid function**.
    * The **final value** of log of odds is $y= w^t x+b$

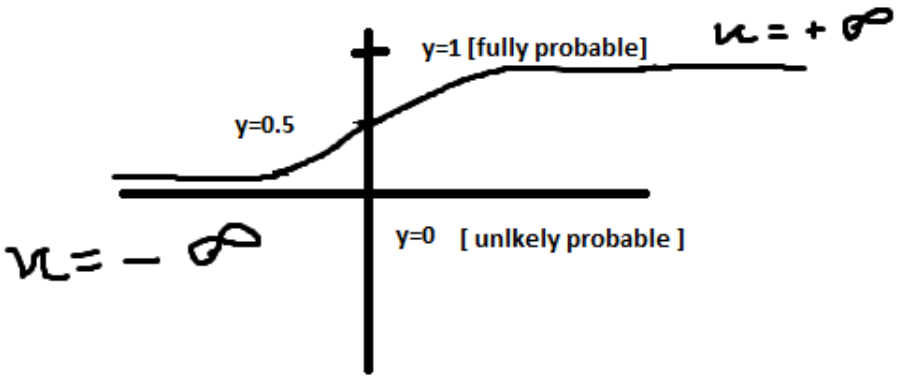
Sigmoid Function (σ)

The sigmoid maps **any real number to [0,1]:**

$$\sigma(z)= \quad 1/(1+e^{-z})$$

- If the value of x tends to **+infinity,** the value of y=1.

- If the value of x tends to **-infinity,** the value of y=0.



LOSS FUNCTION

The loss function (or cost function) in logistic regression measures how well the model's predicted probabilities **match the actual labels**. **Since logistic regression is used for binary classification (outputs y ∈ {0,1})**, it uses a special loss function called **log loss (binary cross-entropy).**

***The loss function penalizes wrong predictions more heavily, especially when the model is confidently wrong.**

* Here, in logistic function, we dont get a middle value with the help of which we would find out loss function.

* For **values of y, we get different values of loss according to the label:**

y=1, ----------> loss = y log $y^{'}$

y=0, ----------> loss = (1-y) log (1-$y^{'}$)

Now the loss function will be,

$$\textbf{Loss function} = y \log y^{'} + (1-y) \log (1-y^{'})$$

here,

y = True label (0,1)

p = predictable probabilty ((σ(z)))

OVERALL ALGORITHM

i) Finding the loss function
ii) **Minimization of error:** Decreasing the value of the loss function by Gradient descent [**by approximation technique**]
iii) **Prediction according to the features provided.**
iv) We will be **returned with a numpy array of weight & biases**.
v) Which when put in the equation, we will get the **lowest value of error**.
vi) **Finally value will be inserted in the sigmoid function.**
vii) RETURNS **ANOTHER NUMPY ARRAY**

CODE [example code with iris dataset petal length]

```
from sklearn import datasets
from sklearn.linear_model import LogisticRegression

iris = datasets.load_iris()

import numpy as np
import matplotlib.pyplot as plt

x = iris["data"][:,3:]

# this means to take all the rows :, will be taken
# this means to take all columns from 3rd to end
# the data set contains i) Sepal Length ii) Sepal Width
                        # iii) Petal Length iv) Petal Width
#so we took petal WIDTH
# x will be a 2D array having all the values of width from data set

y =(iris["target"]==2).astype(np.int64)

# we will search if the target array here is 2 [virginica]
# then it will return True and False if found or not found
# .astype(np.int64) converts the boolearn array into integers (0,1s)


clf=LogisticRegression()
```

```
clf.fit(x,y)

example=clf.predict([[2.6]])

# it takes a 2D array of features and returns predicted class labels
# note that [[]] is given for 2D array
# [[2.6]] is a single sample with petal width 2.6

# model estimates that the probability that a flower is virginica with petal width 2.6

print(example)

#Using matplotlib to plot the visualization taking petal width into consideration

x_new = np.linspace(0,3,1000).reshape(-1,1)

# this part generates 1000 values in betweem 0 to 3
# reshapes 1D array in 2D column vector
# -1 infers the size automatically (here which is 1000)

y_prob = clf.predict_proba(x_new)

# predicting the probabilities of the y axis elements

plt.plot(x_new, y_prob[:,1], "-g", label="virginica")

# here we will print only one line that is the column1 having the 1 (virginica YES)
      #Column 0 (y_prob[:, 0]): Probability of class 0 (not virginica).
      #Column 1 (y_prob[:, 1]): Probability of class 1 (virginica).
# "-g" means "-" solid line style and "g" means green line
# label = "virginica" adds a legend entry for this line

plt.show()
```

SVM (Support Vector Machine)

MAIN POINTS:

* Support vector machine is a **powerful supervised ML algorithm**.
* it is used for **classification** & **regression** tasks.
* It works by **finding the optimal decision boundary** that **best seperates data points of different classes**. [best fit line]

CHARACTERS:

* SVM can handle non-linear classification by mapping input data into higher-dimensional spaces using kernel functions.
* It is **very powerful in high dimensions**.
* It **maximizes the margin between the Classes in SVM i.e** SVM tries to **find the hyperplane** that is **as far away as possible** from **the nearest data points of both classes**.

****MARGIN**: It refers to the **distance between the decision boundary (hyperplane) & the closest data points** from **each class (support vectors)****

****Support Vectors**: Support vectors are the **critical data points** that lie closest to the **decision boundary** in a **Support Vector Machine (SVM)**

* It is a **noise sensitive model**.
  [Distorts the **geometric relations between points affecting kernel computations**]
  [Mislabeled points near the margin **can drastically skew the hyperplane**.]
  [Pulls the **decision boundary toward outliers**, **reducing generalization**.]

DECISION BOUNDARY:

* A decision boundary is a surface (**hyperplane**) that **seperates different classes in a classification problem**.
* It defines the region where the **model predicts one class over another**.
* Helps the model **classify new data points** by **determining** which **side of the boundary they fall on.**

**** Hyperplane:** A hyperplane is a **decision boundary** that seperates **different classes in a dataset**.**

[in 2D space (2 features) it is a line (1D)]
[In 3D space, its a plane (2D)]
[In higher dimension (n-D), its a (n-1) dimensional subspace]

KERNELS IN SVM (Handling Non-Linearity):

What is a kernel?

*****A kernel in SVM is a **mathematical function** that transforms **non-linearly separable data** into a **higher-dimensional space** where it becomes **linearly separable**.

Instead of **fitting complex curves in the original space, SVM uses **kernels** to **compute implicit mappings**, **enabling linear classification** in the transformed space.

* When data is non-linearly seperable in its original space, SVM uses kernel
    i) **Implicitly maps data to a higher dimension space.**
    ii) Finds a **linear hyperline in that new space.**
    iii) Then in the **new space** the data becomes becomes **linearly separable**.
    iv) SVM finds the **optimal hyperplane** there which can **seperate classes**.

$$\text{Nonlinear data} \xrightarrow{\text{Kernel}} \text{High-dimensional space} \xrightarrow{\text{SVM}} \text{Linear separation}$$

How Kernel works (The "kernel Trick") ?

SVM avoids **explicitly computing high dimensional transformations** by using **kernelized dot product**. Kernels compute ϕ(xi) . ϕ(xj) directly without calculating ϕ(x)

NAIVE BAYE'S CLASSIFIER

* A **probablistic machine learning model** based on **Baye's Theorem**.

* It predicts the probability of a class C given input features X:

**** P(A|B) means if B happens then what is the possibility of A happening there****

- P(C) [Prior Probability] = The **baseline probability** of a **class** **before seeing any evidence**.

  -> Reflects general knowledge about how likely the class is without considering specific patient data.

- P(X|C) [Likelihood] =The **probability** of **observing** the **features X** given that the class is C.

  -> How likely is this observation if the patient has the disease?

- P(X) [Evidence] = The **probability** of observing the **features** **across all classes**.

  *-> Ensures P(C|X) is a **valid probability** (between 0 & 1)*

- P(C|X) [Posterior] =The updated propability of the class **after seeing the features X**

  -> Even with a positive test, there's only an 8.7% chance of having the disease (due to the disease's rarity).
  -> Combines prior knowledge P(C) with new evidence P(X|C)

The main formula of Naive Baye's Classifier:

$$P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)}$$

Why Naive?

It assumes that **all feautures are conditionally interdependent** given the class:

p(x1,x2,.....xn|C) = P(x1|C).P(x2|C).....P(xn|C)

This simplifies calculations but is **rarely true in real-world data**. Hence, "NAIVE". The terms money or free might just appear kor random reasons but this model assumes that all of them are **interrelated.**

EXAMPLE

- **Features**: "free" and "money" in an email.
- Compute:
  - $P(\text{Spam}|\text{"free"},\text{"money"}) \propto P(\text{Spam}) \cdot P(\text{"free"}|\text{Spam}) \cdot P(\text{"money"}|\text{Spam})$
  - $P(\text{Not Spam}|\text{"free"},\text{"money"}) \propto P(\text{Not Spam}) \cdot P(\text{"free"}|\text{Not Spam}) \cdot P(\text{"money"}|\text{Not Spam})$
- Predict the class with the larger value.

P(Spam|"free","money")

This expression represents the probability that an email is spam, given that it contains the words "free" and "money".

P(Not Spam|"free","money")
This expression represents the probability that an email is not spam, given that it contains the words "free" and "money".

DECISION TREE

Decision Trees are **supervised learning algorithms**
* used for **classification** and **regression**.
  * They **mimic human decision-making**

HOW?
* by **splitting data into branches** based on **feature values**,
* leading to a **tree-like structure of decisions**.

Key Concepts

Structure of a Decision Tree:

- **Root** Node: The topmost decision point (best feature to split first).
- **Internal** : Decision points based on features.
- **Leaf** Nodes: Final predictions (class labels or continuous values).
- **Branches**: Outcomes of decisions (e.g., "Yes/No" paths)

Working Pattern:

1. **Start at the root** and split the data using the **best feature.**
2. **Repeat** splitting **recursively for child nodes**.
3. **Stop** when a **stopping condition is met**. (max depth, **pure leaf nodes**)

Ensembling Algorithms

1. multiple models (called "weak learners") are combined
2. to produce a stronger, more accurate prediction than any single model alone.
3. It leverages the "wisdom of crowds" principle—aggregating diverse opinions often yields better results.

WHY USE IT?

* Reduces **Overfitting:** Combines models to generalize better.
* **Improves Accuracy**: Averages out biases and errors.
***Handles Complex Data**: Captures nonlinear patterns better than single models.

TYPES OF ENSEMBLING:

1. **Bagging [Bootstrap Aggregating]:** Train multiple versions of the same model on random subsets of data (with replacement) and average their predictions.

   * **Reduces variance**

   ** **variance**[the measurement of **how much a model's predictions fluctuate** when **trained on different subsets** of same **dataset**]**

   ->Single models (especially complex ones like deep trees) can overfit and change wildly with small changes in data.

   -> Bagging **stabilizes** the prediction by **averaging out the noise**.

   ***Doesn't reduce bias much:**

   Since all models are **usually strong** but **overfitting** models, **bagging mainly smooths them**, **not makes them smarter**.

   * Ensembling of **Decision tree**. [combining multiple individual decision trees to create a stronger, more accurate model than any single tree alone.]

   How it works?

i) Randomly pick data points from the training set with **replacement (after picking an item, we can pick it again by putting it back)** **to create a bootstrap sample** (same size as the original dataset, but some points may appear multiple times).

ii) Train a model on this **random sample**.

iii) Repeat (i) & (ii) for many models.

iv) When predicting:

1) Regression: take the average of all model predictions.
2) Classification: do **majority vote**.

2. **Boosting:** Sequentially train models where **each new model corrects errors** of the **previous one**. Focuses on misclassified samples.

How it works?

***REDUCTION OF BIAS [Bias reduction == learning of True complex patterns]**

Bias means the model is too simple to capture the real patterns.

i) In boosting, you **start with weak learners** (**small trees** or **models**) that usually **have high bias** — they **miss important patterns**.

ii) Boosting fixes this by **adding many weak learners sequentially**, where each **new learner focuses** on the **mistakes (errors) of the previous ones**.

iii) By correcting errors little by little, the **model becomes less biased** — it **captures the true structure of the data much better**.

CONS:
* Overfitting
* Slow training

3. **<u>Stacking:</u>** Use of **multiple different models** and **train a meta-model** to **combine** their predictions.
   * Captures the diverse strength of each model.

   How it works?

   i) Train base models on training data.
   ii) Generate predictions for **each base model** but only on **validation data**. [not training data]
   iii) Use **predictions** as **input features** to train a **meta model (level 1)**
   iv) First, **all base models make their predictions**, then the **meta-model uses those predictions to output the final answer**.

   <u>SHORT EXPLAINATION:</u>
   "Stacking = Teamwork. Base models = players. Meta-model = coach who picks the best move based on players' suggestions."

EDA [Explanatory Data Analysis]

Exploratory Data Analysis (EDA) is a critical step in the data analysis process where analysts and data scientists **explore** and **summarize datasets** to **understand their main characteristics**, **patterns**, and **anomalies**. EDA involves using **statistical** and **visualization techniques** to uncover insights **before applying formal modeling or hypothesis testing**.

* Understanding data
* Preprocessing (80% time)
* Modelling (20% time)
* UCA (Understand Clean Analyze)

- **Understanding: [Univariate Analysis]**
  * Understand Data Structure – Identify variables, data types, and distributions.
  * Help in selecting or transforming variables for machine learning.

- Cleaning: [**Anomaly Analysis**]
  ***** No redundancy [repeatation of values avoided]
  * Missing values should be avoided
  * NULL values should not be counted
  * getting rid of all **inconsistencies**
  * Noise (outliers) should not be counted [**extremely high or low values avoidance**]

- **Analyze**: [**Correlation Analysis**]
  *** Exploring relationships between variables**
  * Discover correlations, trends, and associations.
  *Validate assumptions for statistical models (e.g., normality, linearity).
  * Decisions taken can be understood after analyzation

STEPS IN EDA:

1. **Data collection:** Gather raw data from sources & **perform a preliminary check** (CSV, SQL, APIs, etc.).
2. **Data Cleaning**: Handle missing values , Correct inconsistencies & Remove outliers if necessary
3. **Univariate analysis:** Analyse single variables (mean, mode,distribution plot) & Use histograms, box plots, and summary statistics.
4. **Bivariate & Multivariate Analysis:** Explore relationships between variables.
5. **Outlier Detection & Treatment**: Identify and handle anomalies.
6. **Feature Engineering**: Create new features for better analysis.
7. **Final Insights & Reporting**: Summarize key findings.

When To stop EDA:

- When data is **clean & ready for modelling**.
- When **key trends & anomalies** are **identified**.
- When all questions are answered.

PRACTICAL EDA-ing Code

```
import numpy as np
import pandas as pd
import seaborn as sns

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

```
#understand the data

data.head()                # by default shows the first 5 rows info

data.describe()          # for a more statistical approach

data.columns               # shows the column heads of each infos

data.shape                 #shows the dimesion of data

data['Age'].unique     # SHORTLY ALL UNIQUE VALUES WILL BE ON DISPLAY

data.isnull().sum() # returns boolean value
                                # the number of null values of all classes will be shown

new_data = data.drop(['BMI','Pregnancies'],axis=1)
# to get rid of non-necessary values
# new dataset from older dataset
# we will NEVER CHANGE THE OG DATASET
# Always Make copies & use it


#ANALYSIS [strictly requires SEABORN]
corelation = new_data.corr()
sns.heatmap(corelation, xticklabels=corelation.columns, yticklabels=corelation.columns ,annot=True)
      #corelation shows the relation of each data collection with the other

sns.pairplot(new_data)      #pair wise plotting will be done with each info with each

sns.relplot(x='Glucose', y='Age', hue='Outcome',data=data)
#to plot the relations as wanted
#x for x-axis y for y-axos
#hue for the colour

sns.displot(data['Glucose'],bins=5)
# used to create distribution plots,
# which visualize the distribution of a univariate (single-variable) dataset.
# It can generate histograms, kernel density estimates (KDE), and more.
```

UNSUPERVISED MACHINE LEARNING

Clusterring

A clustering model is a type of **unsupervised machine learning model** used to **group similar data points together based on their features.**

* Unlike supervised learning, clusterring does not rely on **predefined labels.**
* It **identifies patterns & structures** in data.
* by **measuring similarity or distance between point**s.

Main concepts:

- **Unsupervised Learning:** No labeled output is provided; the model finds hidden patterns.
- **Grouping by Similarity:** Data points in the same cluster are more similar to each other than to those in other clusters.
- **Distance Metrics:** Common measures include Euclidean distance, Manhattan distance, and cosine similarity.
- **No Single "Correct" Solution:** Different algorithms may produce different clusters based on parameters.

There are multiple types of alogrithm

K-Means [we know the number of clusters]:

* K-Means is one of the **most popular centroid-based clustering algorithms** in unsupervised machine learning.

* It partitions data into **K (known number) clusters**, where **each cluster** is represented by its **centroid (the mean of points in the cluster)**.

How it works?

1. Choose the Number of Clusters (K)

* The user must specify the numbers of clusters (K) [number of groups]

methods of choosing K:

i) *Elbow method*: look for the elbow in the WCSS (Within-Cluster Sum of Squares) plot .

****WCSS** is a **key metric** used in **K-Means clustering** to **measure how compact (tightly grouped)** the **clusters are**. It quantifies the **total squared distance** between **each data point** and **its assigned cluster centroid**.**

Lower WCSS==Compact cluster
Higher WCSS==Poor Clustering

****Elbow point** is the point where WCSS **stops decreasing sharply** suggests the best **k****

ii) *Silhouette Score*: The Silhouette Score is a metric used to **evaluate the quality of clustering algorithms**. Unlike WCSS (Within-Cluster Sum of Squares), which only measures cluster compactness, the Silhouette Score considers both:

[1] **How well seperated clusters are (**distance between clusters**)**

[2] **How tightly grouped points are within their own cluster (**cohesion**).**

2. ## Initialize Centroids Randomly

* Randomly pick K data points as initial centroids.

3. ## Assign points to Nearest Centroid

* For each point, calculate its distance (usually Euclidean distance) to all centroids.
*Assign the point to the nearest centroid, forming K clusters.

4. ## Update Centroids

*Recompute the centroid of each cluster as the mean of all points in that cluster.

5. ## Repeat Until Convergence

* Repeat step 3 & 4 until:
    i) centroid no longer changes significantly.
    ii) maximum number of iterations reached.

Hierarchial Cluster

Hierarchical clustering is an unsupervised clustering algorithm that builds a **tree-like structure (dendrogram)** to **group similar data points**.

*it does not require pre-specifying the number of clusters (K).
* [if in tree based form groups made in nodes and sub nodes]

How it works?

1. **Compute Distance Matrix:**
   * Calculate pairwise distances (Euclidean, Manhattan, Cosine, etc.) between all points.

2. **Merge Closest Cluster:**
   Use a linkage criterion to decide how to merge clusters:

   * Single Linkage: Minimum distance between clusters.
   * Complete Linkage: Maximum distance between clusters.
   * Average Linkage: Mean distance between clusters.
   * Ward's Method: Minimizes variance (like K-Means).

3. **Update Distance Matrix**:
   Recompute distances after merging.

4. **Repeat Until One Cluster Remains:**
   * Continue until all points are in a single cluster.

5. **Choose Cluster via Dendrogram**:
   * Cut the dendrogram at a desired height to get clusters.

- DB scan [density-based **closest data points are taken together**]
  *

- Optical [Advanced dbscan plus distance measurement done too]

1, DB Clusterring
2, Naive Bayes
4, Logisitc Regression
5, Linear Regression
7, K Nearest
25, Random Forest Algo
30, K-Means Algorithm
33, Decision

Why AI?

* Improved efficiency
* The simulation of human intelligence inside machines
* It alone can operate , make decisions, reason etc. [Automated Human cognitive abilities]

Why ML?

* **Machine Learning (ML) is the ability given to AI systems to learn from data and improve their performance without being explicitly programmed for every scenario.**

* It helps **AI adapts and evolves** (e.g., AlphaZero taught itself chess in 4 hours and beat world champions).

* **Without ML**: AI relies on **rigid**, **pre-programmed rules** (e.g., 1990s chess bots).

* Basically instead of **hard-coding rules**, we train AI using data.

* Focuses on algorithms that improve automatically through experience (e.g., decision trees, SVM, regression).

EXAMPLE:

Traditional Programming [without ML]: Manually **write rules** to **filter spam emails** (e.g., "block emails with 'FREE' in the subject"). Hardcoded approach

ML Approach: Show the AI thousands of spam/non-spam emails, and **let it learn patterns on its own**.

Why DL?

*Feature extraction: The process of **identifying** & **isolating meaningful patterns** from **raw data** to **make it easier for the model to analyze** as raw data is hard for the machine to analyze from or process directly. **Features** simplify data while **preserving important information**.

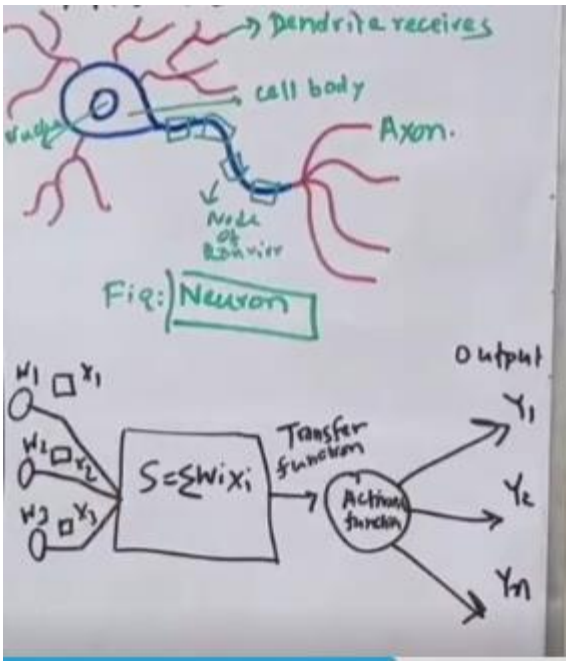
*Object Detection: Object detection is the **task of identifying** and **locating objects** within an image/video, typically by **drawing bounding boxes** around them.

***With ML:**

For feature extraction, we need human intervention. We need to change the activities in training testing & supervision. We need to check individually the different types of factors like overfitting, underfitting etc.

***With DL**: This feature extraction is done automatically **requiring low human intervention** .

NEURAL NETWORK



This image draws a powerful parallel between **biological neurons** in the human brain and **artificial neurons** in **deep learning (DL)**. Let's break it down using the picture:

🧠 Biological Neuron

- ***Dendrites***: Receive signals from other neurons.
- ***Cell Body*** (Soma): Processes the input signals.
- ***Axon***: Transmits the output signal to other neurons.
- ***Node of Ranvier***: Speeds up signal transmission.

🤖 Artificial Neuron (in a Neural Network)

- **Inputs** ($x_1, x_2, ..., x_n$): **Analogous to signals received by dendrites**.
- **Weights** ($w_1, w_2, ..., w_n$): **Represent the strength of each input signal** (like synapse strength).
- **Summation Function** ($S = \Sigma\, w_i x_i$): Just like the **cell body processes inputs**, the **weighted sum combines input signals**.
- **Activation Function**: **Mimics the firing of a neuron**. It **decides** whether the **neuron "fires" (sends output)** **based on the input sum**.
- Outputs ($y_1, y_2, ..., y_n$): Like **signals sent down the axon** to other neurons in the network.

🔁 Relevance in Deep Learning:

* Neural networks are **inspired by biological neurons**.

* A **deep neural network** is like **a large web of interconnected artificial neurons passing data and learning from it**.

* The **learning** in **DL** involves **adjusting the weights** (w_i) so the **output matches desired values** (similar to how **biological synapses strengthen** or **weaken over time with learning**).