Page No.
Date 23 10 24

MACHINE LEARNING

ML Techniques
- Supervised ML
- Unsupervised ML
- Reinforcement ML

Supervised ML :→ We have labeled data
→ Predict House price
→ Two types of Problem Statement → Regression (continous) / Classification (catergorical)
→ Divides the feature in two categories
① Dependent features
② Independent features

Eg:

size of house	No. of rooms	Price
5000	5	450 k
6000	6	500 k

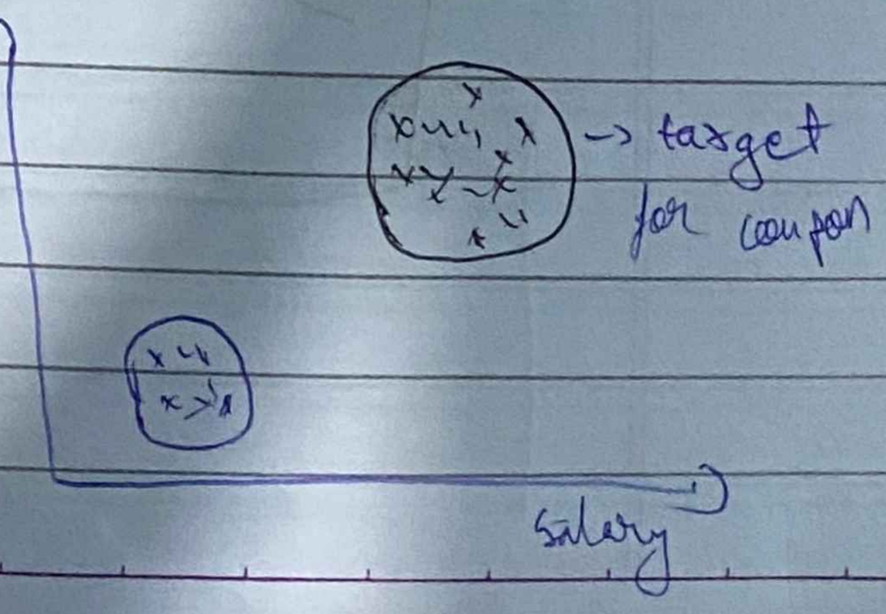
} continous

In Supervised ML we have
① Dependent or O/P feature (Mandatory
② If the dependent feature → continous → Regression / → Categorical → classification.

Unsupervised ML
In this we dont predict anything
we make cluster of unlabeled data

Eg:

Salary	Spending Score (1-10)
20000	1
4500	2



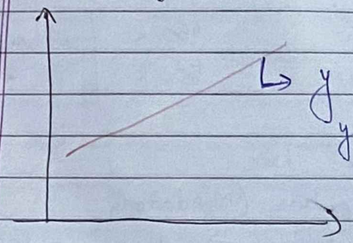
All Thanks To Krish Sir!

Date

Algorithms we have to learn

Supervised ML	Unsupervised ML
① Linear Regression	① K means
② Ridge & Lasso	② Hierarchical Mean
③ Elastic Net	③ DBSCAN clustering.
④ Logistic Regression (classification)	
⑤ Decision Tree	
⑥ Random Forest] Both classification & Regression	
⑦ Ada Boost	
⑧ Xgboost	

Equation of Line, 3D plane & Hyperplan (in Dimension)



$\hookrightarrow y = \underbrace{m}_{\text{slope}}x + \underbrace{c}_{\text{intercept}}$

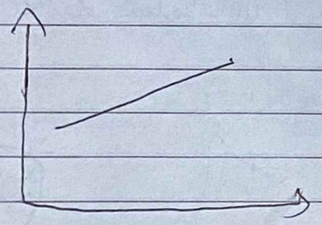
$y = \beta_0 + \beta_1 x$

$ax + by + C = 0$ }

$by + C = -ax$

$by = -ax - C$

$y = \underbrace{\boxed{\frac{-a}{b}}}_{m} x - \underbrace{\boxed{\frac{C}{b}}}_{C}$



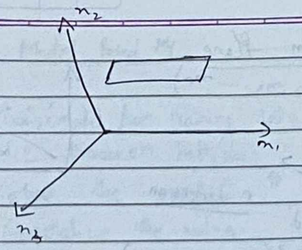
$w_1x_1 + w_2x_2 + b = 0$

$\boxed{w^Tx + b = 0}$

⇓

Eq of a straight line.

Page No.
Date



$w_1x_1 + w_2x_2 + w_3x_3 + b = 0$

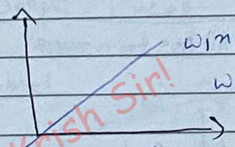
$\boxed{w^Tx + b = 0}$

$w = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$

n - Dimension plane :

$w_1x_1 + w_2x_2 + w_3x_3 \ldots\ldots w_nx_n + b = 0$

$\boxed{w^Tx + b = 0}$



$w_1x_1 + w_2x_2 + b = 0$

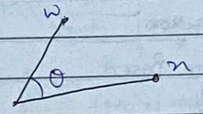
$w_1x_1 + w_2x_2 = 0$

w^Tx

Eqn of a straight line passing through origin $\boxed{w^Tx = 0}$

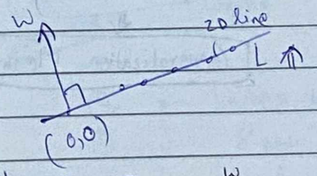
Eqn. of a plane = ⇑n : $w^Tx = 0$

$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_n \end{bmatrix} \quad \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$



$w^Tx = 0$

$w \cdot x = w^Tx = \|w\| \|x\| \cos\theta = 0$

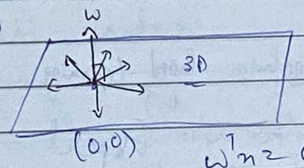


$\theta = 90^\circ$

$\cos\theta = 0$

$\boxed{w \perp ⇑}$

intercept = 0

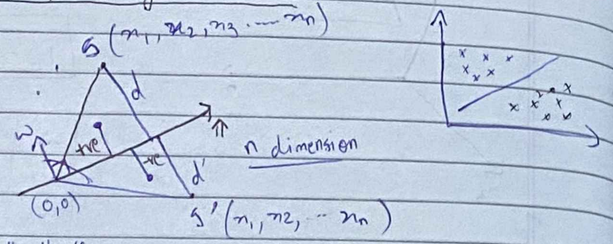


$w^Tx = 0$

All Thanks To Krish Sir!

Distance of a point from Plane :

$\boxed{w^T x = 0}$



$\Rightarrow w^T S = \|w\| \|S\| \cos\theta = +ve$

θ = b/w 0 & 90

$$\boxed{d = \frac{w^T S}{\|w\|}}$$

$$\boxed{d = \frac{w^T S'}{\|w\|}} \Rightarrow w^T S' = \|w\| \|S\| \cos = -ve$$

Instance Based Learning Vs Model Based Learning

Learning → memorising / Generalizing

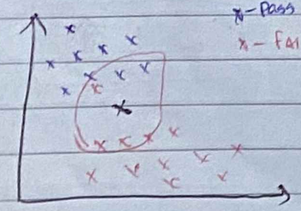
Use case → ML Model → Regression / classification

Learning Pattern

Instance Based → Data → O/P

Model Based Technique → DATA → Pattern → Generalization Method

No. of play | No. of study hours | Pass/Fail



Instance Based learning ⇓ Domain Expert

Model Based ML	Instance Based ML
1) Train model from training data i.e Discover Patterns	1) Do not train Model No pattern discovering
2) store the model	2) There is no model to store
3) Generalize the rules	3) No generalization for scoring
4) Can throw away input/training data after Model training	4) Input/training Data must be kept
5) Requires less storage	5) Requires more storage
6) Fast process	6) slow process

Simple Linear Regression :

supervised ML → Regression

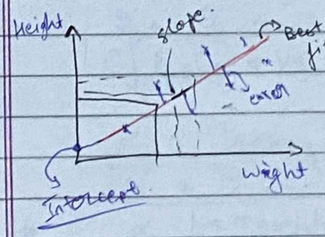
weight → Model → Height

Dataset

Weight (independent)	Height (dependent)
74	170 cm
80	150 cm
75	175.5 cm

simple Linear Regression: When we have just one input feature we call it simple linear regression

Multiple linear Regression : When we have more than one input feature.



(Minimal) error

$y = mx + c$

$y = \beta_0 + \beta_1 x$

x refers to independent feature.

$$\boxed{h_\theta(x) = \theta_0 + \theta_1 x}$$

θ_0 = Intercept

θ_1 = Slope or coefficient

for multi feature

if $x = 0$
$h_\theta(x) = \theta_0$
$h_\theta(x) = \hat{y}$

$$\boxed{h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_n x_n}$$

$$\boxed{\text{Error} = y - \hat{y}}$$

All Thanks To Krish Sir!

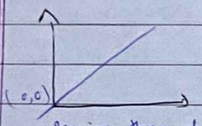
Cost Function :

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(\underbrace{h_\theta(x)^{(i)}}_{\text{predicted point}} - \underbrace{y^{(i)}}_{\text{True value}} \right)^2$$

mean squared Error

Final Aim (What we need to solve)

Minimize $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x)^{(i)} - y^{(i)} \right)^2$



if $\theta_0 = 0$, $h_\theta(x) = \theta_1 x$

passing through origin

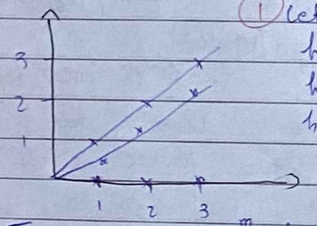
Dataset	x	y
	1	1
	2	2
	3	3

(1) let $\theta_1 = 1$ {slope}

$h_\theta(x) = 1$, $x = 1$

$h_\theta(x) = 2$, $x = 2$

$h_\theta(x) = 3$, $x = 3$



$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x)^{(i)} - y^{(i)} \right)^2$$

$$= \frac{1}{2 \times 3} \left[(1-1)^2 + (2-2)^2 + (3-3)^2 \right]$$

$$= 0$$

(2) let say $\theta_1 = 0.5$

$h_\theta(x) = 0.5$ if $x = 1$

$h_\theta(x) = 1$ $x = 2$

$h_\theta(x) = 1.5$ $x = 3$

$$J(\theta_1) = \frac{1}{2 \times 3} \left((0.5-1)^2 + (1-2)^2 + (1.5-3)^2 \right)$$

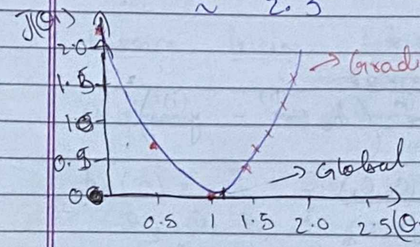
$$= \frac{1}{6} \left(0.25 + 1 + 2.25 \right)$$

$$= \approx 0.58$$

(3) if $\theta_1 = 0$

$$J(\theta_1) = \frac{1}{2 \times 3} \left((0-1)^2 + (0-2)^2 + (0-3)^2 \right)$$

$$\approx 2.3$$



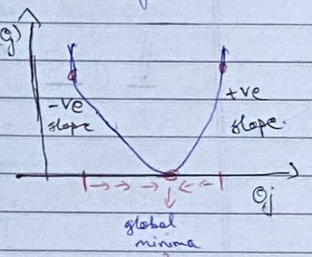
Convergence Algorithm: {optimize the changes of θ_1 value}

Repeat until convergence

{

$$\theta_j := \theta_j - \alpha \frac{d\, J(\theta_j)}{d\, \theta_j}$$

}



$\theta_j = \theta_j - \alpha(-ve)$ → $\theta_j = \theta_j + \alpha$

$\theta_j = \theta_j - \alpha(+ve)$ → $\theta_j = \theta_j - \alpha$

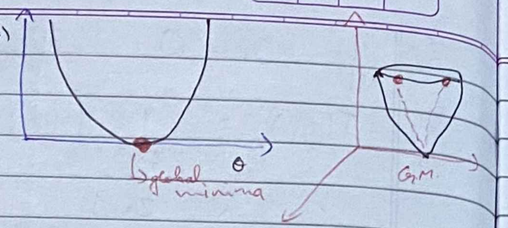
α = Learning Rate (controls the convergence rate)

$\boxed{\alpha = 0.0001}$

if α is greater then to reach global minima is difficult, it may reach from -ve slope to +ve slope & vice-versa

All Thanks To Krish Sir!

Final conclusion :



Gradient Descent

Convergence Algorithm :

repeat until convergence

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x)^{(i)} - y^{(i)} \right)^2$$

$$\left\{ \theta_j := \theta_j - \alpha \frac{d\, J(\theta_0, \theta_1)}{d\, \theta_j} \right\}$$

$$\frac{d\, J(\theta_0, \theta_1)}{d\, \theta_j} = \frac{d}{d\theta_j} \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x)^{(i)} - y^{(i)} \right)^2$$

if $j = 0$:

$$\Rightarrow \frac{d\, J(\theta_0, \theta_1)}{d\theta_0} = \frac{d}{d\theta_0} \frac{1}{2m} \left[\sum_{i=1}^{m} \left(h_\theta(x)^{(i)} - y^{(i)} \right)^2 \right]$$

$$= \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x)^{(i)} - y^{(i)} \right) \times 1$$

$$= \frac{1}{m} \quad h_\theta(x) = \theta_0 + \boxed{\theta_1 x} \rightarrow 0$$

if $j = 1$:

$$\Rightarrow \frac{d\, J(\theta_0, \theta_1)}{d\theta_1} = \frac{d}{d\theta_1} \frac{1}{2m} \left[\sum_{i=1}^{m} \left((\theta_0 + \theta_1 x) - y^{(i)} \right)^2 \right]$$

$$= \frac{1}{m} \sum_{i=1}^{m} \left((\theta_0 + \theta_1 x) - y^{(i)} \right) (x)$$

Repeat until convergence :

$$\left\{ \theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x)^{(i)} - y^{(i)} \right) \right.$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x)^{(i)} - y^{(i)} \right) x^{(i)}$$

All Thanks To Krish Sir!

Multiple Linear Regression

Dataset

I/P	O/P feature
Weight	Height

$h_\theta(x) = \theta_0 + \theta_1 x$ (I/P or independent)

θ_0 = intercept

θ_1 = slope

Housing Pricing Dataset:

Independent			O/P feature
No. of rooms	size of House	Location	Price

$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$ {Multiple linear regression}

$\theta_1, \theta_2, \theta_3$ = coefficient

θ_0 = Intercept

Performance Metrics Used In Linear Regression

① R - Squared

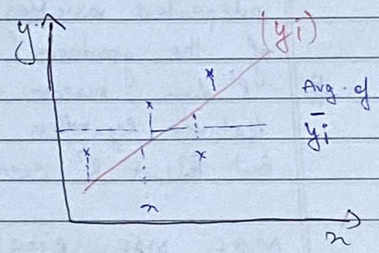
$$R\text{-squared} = 1 - \frac{SS_{Res}}{SS_{Total}}$$

$$= 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y}_i)^2}$$

$$= 1 - \frac{\text{Small no.}}{\text{Bigger no}}$$

$$= 1 - \text{small no}$$

$$\approx 1$$



0.70 => 70%

0.85 => 85%

0.90 => 90%

more towards 1, more accurate model.

② Adjusted R squared

Size of house ↑↑ Price ↑↑

+ve - correlation

Data set - (Price)

Gender: size of house → Price No. of Bedroom location

R-squared = 75% ⇒ 0.75

" = 80% ⇒ 0.80

" " = 85% ⇒ 0.85

" " = 87% ⇒ 0.87

$$\text{Adjusted } R^2 = 1 - \frac{(1-R)^2 (N-1)}{N-P-1}$$

N = No. of data points

P = No. of Independent features

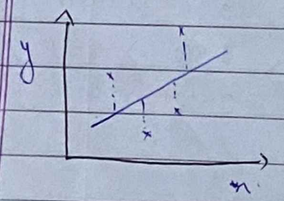
P=2, R^2=90%. R^2-adjusted = 86%.

P=3, R^2=92%. R^2-adjusted = 82%.

It adjusts the R-squared value removing the unnecessary independent variables & provides a more accurate measure of the goodness of fit.

R^2 always increase or stay the same, when you add more features. Regardless of whether those predictor are relevant. But Adjust R^2 even decreases it.

MSE, MAE, RMSE → Cost function:



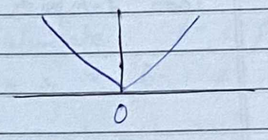
Mean squarred error = $\sum_{i=1}^{n} \frac{(y-\hat{y})^2}{n}$ ⇒ cost func. ↓↓

↳ Quadratic equation

Advantage	Disadvantages
① Differentiable	① Not Robust to olliers (Do not penalizes outliers)
② It has one local minima & one global minima	② It is no longer in same units.
③ Converges faster	

Mean Absolute Error (MAE):

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



Advantage	Disadvantage
1. Robust to outliers	1. Convergence usually take more time. optimization is a complex task.
2. It will be in the same unit	2. Time consuming

Root Mean Squared Error (RMSE):

$$RMSE = \sqrt{MSE}$$

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

Advantage	Disadvantage
1. same unit	1. Not Robust to outliers
2. Differentiable	

As a fresher use

MSE
RMSE
MAE
R^2
Adj - R^2

⇒ makes performance metrice

All Thanks To Krish Sir!

Page No.
Date 27 10 24

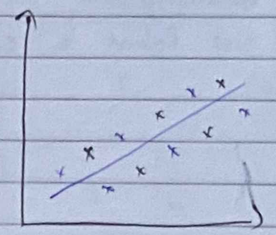
Overfitting & Underfitting (Bias & variance)

1. Training dataset
2. Test dataset
3. Validation dataset.

Dataset:
1000 datapoints → Training Dataset (700) → training the model
→ Test Dataset (300) → testing the model

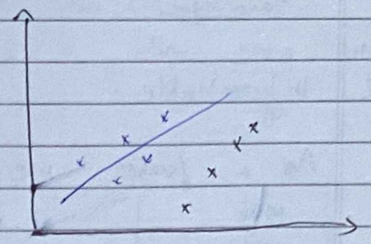
Training Dataset:
- Train → Train the model
- Validation → Hyperparameter tunning your model

	Generalized	Overfitting	Underfitting
Train:	Very good Accuracy [Low Bias]	Very good Accuracy (90%) (Low Bias)	Bad Accuracy (High Bias)
Test:	V. good Accuracy [Low variance]	Bad Accuracy (50%) (High variance)	Bad Accuracy (High Bias)



Generalized Model

Low Bias, Low variance



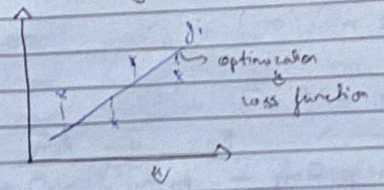
Overfitting

Low Bias, High variance.

$\frac{2}{n_1}\frac{n}{\beta_1} \rightarrow n_1$

Page No.
Date

Linear Regression Using OLS (ordinary Least Square)



OLS → Formula and Calculate β_0 & β_1

Ordinary Least square $h_\theta(x) = \hat{y} = \beta_0 + \beta_1 x_1$

$$S(\beta_0, \beta_1) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_1)^2$$

find β_0 & β_1

$$\frac{d S(\beta_0, \beta_1)}{d\beta_0} = \frac{2}{n}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_1)(0 - 1 - 0)$$

$$= \frac{-2}{n}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_1) = 0 \rightarrow (1)$$

$$\frac{d S(\beta_0, \beta_1)}{d\beta_1} = \frac{2}{n}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_1)(-x_1) = 0$$

$$= \frac{-2}{n}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_1)(x_1) = 0 \rightarrow (2)$$

Eqn (1)
$$\frac{-2}{n}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i) = 0$$

$$-\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i) = 0$$

$$-\sum_{i=1}^{n} y_i + n\beta_0 + \beta_1\sum_{i=1}^{n} x_i = 0$$

$$n\beta_0 + \beta_1\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$$

$$\beta_0 = \frac{\sum_{i=1}^{n} y_i}{n} - \beta_1\frac{\sum_{i=1}^{n} x_i}{n} \Rightarrow \boxed{\beta_0 = \bar{y} - \beta_1\bar{x}}$$

→ Intercept

All Thanks To Krish Sir

we can multiply 2 to step directly β_0

Eqn: 2: $-\frac{2}{n}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)(x_i) = 0$

$$\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)(x_i) = 0$$

$$\sum_{i=1}^{n}(x_i y_i - \beta_0 x_i - \beta_1 x_i^2) = 0$$

$\beta_0 = \bar{y} - \beta_1 \bar{x}$

$$\sum_{i=1}^{n}(x_i y_i - x_i\bar{y} + \beta_1 \bar{x} x_i - \beta_1 x_i^2) = 0$$

$$\sum_{i=1}^{n}(y_i - \bar{y} + \beta_1 \bar{x} - \beta_1 x_i)\, x_i = 0$$

$$\sum_{i=1}^{n}(y_i - \bar{y}) + \beta_1(\bar{x} - x_i) = 0$$

$$\sum_{i=1}^{n}\beta_1(\bar{x} - x_i) = -\sum_{i=1}^{n}(y_i - \bar{y})$$

$$\beta_1 = \frac{-\sum_{i=1}^{n}(y_i - \bar{y})}{\sum_{i=1}^{n}(\bar{x} - x_i)}$$

$$\boxed{\beta_1 = \frac{+\sum_{i=1}^{n}(y_i - \bar{y})}{+\sum_{i=1}^{n}(x_i - \bar{x})}}$$

OLS

$$\boxed{\beta_0 = \bar{y} - \beta_1 \bar{x}} \qquad \boxed{\beta_1 = \frac{\sum_{i=1}^{n}(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})}}$$

x	y	$(y_i - \bar{y})$	$(x_i - \bar{x})$	β_1	$\beta_0 (\bar{y} - \beta_1 \bar{x})$

2-5:

Practical Simple Linear Regression:

```
import pandas as pd
impot Numpy as np
import matplotlib.pyplot as plt
```

Independent & dependent features

```
X = df[['Weight']]   ## independent features should be data frame or 2 dimensional array
Y = df['Height']     ## For dependent feature it can be in series or 1D array
```

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=0.25, random_state=42)
```

Standardization (Only done on independent feature or input feature)

$\mu = 0$, $\sigma = 1$

$z\text{-score} = \frac{x_i - \mu}{\sigma}$

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Only technique needs to be transferred not data
So the test data is not seen by model.
to avoid data leakage.

All Thanks To Krish Sir!

Apply Simple linear regression :

```
from sklearn.linear_model import LinearRegression
regression = LinearRegression (n_jobs = -1)
regression.fit (X_train, y_train)
```
↳ LinearRegression (n_jobs = -1)

```
Print ("Coefficient or slope:", regression.coef_)
Print ("Intercept:", regression.intercept_)
```
↳ $y(\theta) = \beta_0 + \beta_1 x_1$ (intercept, slope)

↳ Coefficient or slope : [17.2982057]

↳ Intercept : [156.47058823529412]

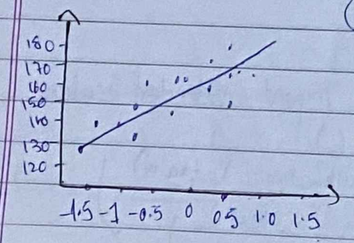
Imp.

Coeff. : means one unit movement in weight value that leads to 17.29 unit movement in height value

Intercept : means when X axis (weight) is zero, where your y-axis is at 156.47.

plot Training data plot best fit line

```
plt.scatter (X_train, y_train)
plt.plot (X_train, regression.predict (X_train))
```



Prediction of test-data

1. predicted height output = intercept + coeff. (weights)
2. y_pred_test = 156.47 + 17.29 (X_test)

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

Performance Metrics

```
from sklearn.metrics import mean_absolute_error, mean_squared_error
mse = mean_squared_error (y_test, y_pred)
mae = mean_absolute_error (y_test, y_pred)
rmse = np.sqrt (mse)
print (mse)  →  114.54
print (mae)  →  9.66
print (rmse) →  10.71
```

R-square

```
from sklearn.metrics import r2_score
score = r2_score (y_test, y_pred)
print (score) → 0.7360
```

display adjusted R-squared

```
1 - (1 - score) * (len (y_test) - 1) / (len (y_test) - X_test.shape[1] - 1)
```
↳ 0.670

OLS Linear Regression

```
import statsmodels.api as sm
model = sm.OLS (y_train, X_train).fit()
prediction = model.predict (X_test)
print (prediction)
[5.794  5.794  -28.78  23.6  -7.8  34.08 ]
Print (model.summary())
```

All Thanks To Krish Sir!

Assumptions:
1) Plot a scatter plot for prediction
plt.scatter (y_test, y_pred)
2) plot residuals
3) scatter plot with respect to residuals & prediction.

Page No.
Date 28 10 24

Prediction for new-data
regression.predict(scaler.transform([[72]]))

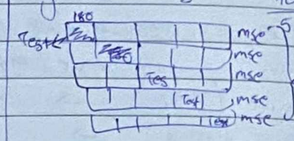
↳ array([155.97744705])

1-9:

Multiple Linear Regression - Economics Dataset

1000 Records
↙ ↘
Train (900) Test (100)
↙ ↘
Train Validation

if cv = 5. 900 = 180.



What is Cross Validation:
Technique to assess how well a model generalizes to an unseen data

What is scoring:
scoring is the metric used to evaluate how well the model is done in each split.
In regression problem scoring are RMSE, MSE, MAE

independent & dependent features

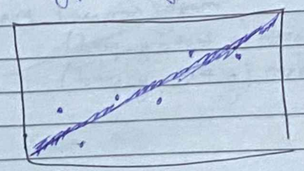
X = df_indep.iloc[:, :-1]
y = df_indep.iloc[:, -1]

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=0.25, random_state=42)

Page No.
Date

import seaborn as sns

sns.regplot(df_indep['interest_rate'], df_indep['index_price'])



# from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)

# from sklearn.linear_model import LinearRegression
regression = LinearRegression()
regression.fit(X_train, y_train)

# from sklearn.model_selection import cross_val_score
validation_score = cross_val_score(regression, X_train, y_train, scoring='neg_mean_squared_error', cv = 3)

np.mean(validation_score)

prediction
y_pred = regression.predict(X_test)
y_pred

↳ array([118.0.

All Thanks To Krish Sir

Performance Metrics :

```
from sklearn.metrics import mean_absolute_error,
                            mean_squared_error
mse = mean_squared_error (y_test, y_pred)
mae = mean_absolute_error (y_test, y_pred)
rmse = np.sqrt (mse)
print (mse)
print (rmse)
print (mae)

from sklearn.metrics import r2_score
score = r2_score (y_test, y_pred)
print (score)
# display adjusted R-squared.
print (1 - (1-score) * (len (y_test) - 1) / (len (y_test) - X_test.shape [1] - 1))
```

(Test truth data) (Test predicted data)

Assumptions :

```
residuals = y_test - y_pred
print (residuals)
## Plot this residual
sns.displot (residuals, kind = 'kde')
plt.scatter (y_pred, residuals)
```

residual plot should be normally distributed

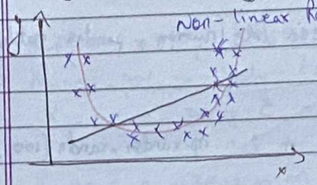
OLS Linear Regression :

```
import statsmodels.api as sm
model = sm.OLS (y_train, X_train).fit()
print (regression.coef_)
↳ [88.2 , -116.27]
```

coeff. in summary & actual are same.

All Thanks To Krish Sir!

Polynomial Regression :



Non-linear Relationship

$h_\theta(x) = \beta_0 + \beta_1 x \rightarrow$ simple Linear Regression

$h_\theta(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$

↳ Multiple Linear regression.

Error ↑↑ , Error ↓↓ → Polynomial Regression.

Polynomial Degrees :

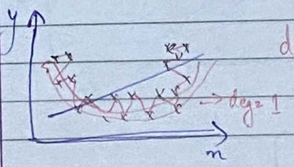
simple Polynomial Regression {1 I/P & 1 O/P}

polynomial degree = 0 , $h_\theta(x) = \beta_0 \times x^0$ ⇒ constant value

polynomial degree = 1 , $h_\theta(x) = \beta_0 \times x^0 + \beta_1 \times x^{(1)}$ ⇒ simple linear reg.

" " degree = 2 , $h_\theta(x) = \beta_0 \times x^0 + \beta_1 \times x^1 + \beta_2 \times x^{(2)}$

$$\boxed{\text{Polynomial degree} = n \quad h_\theta(x) = \beta_0 x^0 + \beta_1 x^1 + \beta_2 x^2 + \beta_3 x^{(3)} \ldots\ldots \beta_n x^{(n)}}$$



degree = 15 (High) It starts to overfit the best fit line.

Multiple ~~linear~~ Independent feature.

degree = 1 : $h_\theta(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2$

degree = 2 : $h_\theta(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1^2 + \beta_4 x_2^2$

Polynomial Regression – Implementation

→ import all imp libraries like (numpy, pandas, matplotlib)

```
x = 6 * np.random.rand(100,1) - 3
y = 0.5 * x**2 + 1.5 * x + 2 + np.random.randn(100,1)
```

$y = 0.5x^2 + 1.5x + 2 +$ outliers

```
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=2, include_bias=True)
x_train_poly = poly.fit_transform(x_train)
x_test_poly = poly.transform(x_test)

from sklearn.metrics import r2_score
regression = LinearRegression()
regression.fit(x_train_poly, y_train)
y_pred = regression.predict(x_test_poly)
score = r2_score(y_test, y_pred)
print(score)
```

↳ 0.9393

```
print(regression.coef_)
```

↳ [[0. 1.5 0.5]]

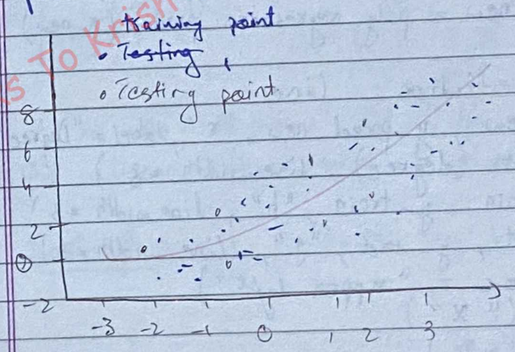
β_0 β_1 β_2

```
print(regression.intercept_)
```

↳ [1.8467]

Prediction of new data set

```
x_new = np.linspace(-3, 3, 200).reshape(200, 1)
x_new_poly = poly.transform(x_new)
x_new_poly

y_new = regression.predict(x_new_poly)
plt.plot(x_new, y_new, "r-", linewidth=2, label="Predictions")
plt.plot(x_train, y_train, "b.", label='Training points')
plt.plot(x_test, y_test, "g.", label='Testing points')
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()
```



All Thanks To Krish Sir!

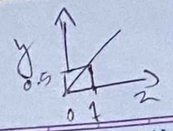
series of steps applied to your dataset in a specific order
① Generating polynomial feature
② Scaling
③ modeling

Pipeline in Polynomial :

```
from sklearn.pipeline import Pipeline

def poly_regression (degree) :
    x_new = np.linspace (-3, 3, 200).reshape (200, 1)

    poly_features = PolynomialFeatures (degree = degree, include_bias
                                        = True)

    lin_reg = LinearRegression ()
    poly_regression = Pipeline ([
            ("poly_features", poly_features),
            ("lin_reg", lin_reg)  ])
    poly_regression.fit (x_train, y_train)
    y_pred_new = poly_regression.predict (x_new)

# plotting prediction line
plt.plot (x_new, y_pred_new, 'r', label = "Degree" +
          str (degree), linewidth = 3 )
plt.plot (x_train, y_train, "b.", linewidth = 3)
plt.plot (x_test, y_test, "g", linewidth = 3)
plt.legend ( loc = "upper left")
plt.xlabel (" x ")
plt.ylabel (" y ")
plt.axis ([-4, 4, 0, 10])
plt.show ()

poly_regression ( 1 )
```

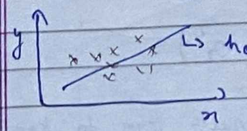
degree.

All Thanks To Krish Sir!



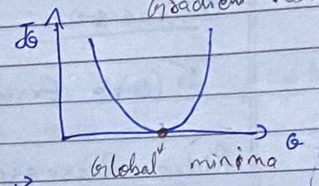
① Ridge Regression (L_2 Regularization) → Reduce overfitting

up till now what we have learned
Linear Regression



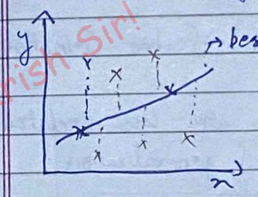
↳ $h_\theta(x) = \theta_0 + \theta_1 x$

Gradient Descent



Cost fn $= \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x)^i - y^i)^2$

In Ridge Regression :



Overfitting
Train data → Acc ↑ → Low Bias
Test data → Acc ↓ → High Variance.

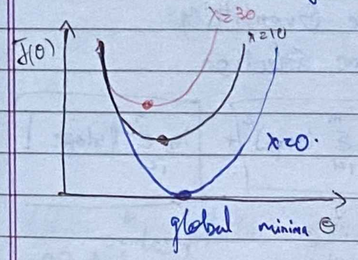
Cost fn : $\frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x)^i - y^i)^2$

It is the Algorithm that helps us hyperparameter tune the linear regression. In short it reduce overfitting.

Cost fn : $\frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x)^i - y^i)^2 + \boxed{\lambda = \sum_{i=1}^{n} (\text{slope})^2}$

↳ hyperparameter
$\lambda \uparrow$, slope $\downarrow$

$\lambda = 1$ if $= 0 + (1)[(\theta_1)^2]$
then cost fn > 0.



$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$
$$= 0.34 + 0.57x_1 + 0.48x_2 + 0.24x_3$$
$$= 0.34 + 0.40x_1 + 0.38x_2 + 0.14x_3$$

conclusion :

$$J(\theta) = \text{Error Term} + \lambda(\theta_1^2 + \theta_2^2)$$

Ridge Regression: Introduces a penalty term to keep θ_1 small, prevent overfitting & improving model generalization.

② Lasso Regression (L_1 Regularization) → Feature Selection.

→ It removes the feature which are less correlated and as a result improves Model generalization.

$$\text{Cost fn}: \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x)^i - y^i) + \boxed{\lambda \sum_{i=1}^{n}(\text{slope})}$$

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$
$$h_\theta(x) = 0.52 + 0.65x_1 + 0.72x_2 + 0.54x_3 + \underline{0.12x_4}$$

↓

Lasso Regression.

$$= 0.52 + 0.65x_1 + 0.72x_2 + 0.54x_3 + \boxed{\cancel{0.12x_4}} \to 0$$

③ Elastic Regression → i) Reduce Overfitting
→ ii) Feature Selection

$$\text{Cost fn}: \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x)^i - y^{(i)})^2 + \boxed{\lambda_1 \sum_{i=1}^{m}(\text{slope})^2} + \boxed{\lambda_2 \sum_{i=1}^{m}|\text{slope}|}$$

↓ Reduce Overfitting ↓ Feature Selection

Types of Cross Validation :

Data set → 70% → Train, 30% → Test

Train → Train, Validation ← cross validati: For doing the split of training ← Random state.

① Leave one out CV (LOOCV)

Training → 500 Records. ↑↑ Complexity of Traing Model.

Exp 1 — Training → Acc-1
validation

Exp 2 — Tra | ining → Acc-2

Exp 500 — Training → Acc-500.

Disadvantages → ① Overfitting → Training ↑↑ Acc, Validation ↓↓ Acc → New Test data → Acc ↓↓

② When large records complexity of training model.

② Leave P out CV (P = 20, P = 30)

③ k-fold CV

K = 5, n = 500 $\quad \frac{n}{k} = \frac{500}{5} = 100$

CV = 1, Exp: 1 — Training → Acc-1

Exp: 2 → Acc-2

⋮

Exp: 5 → Acc-5

Training, Validation

Avg. Accuracies

① Splits the data into k part (eg: k = 5)

② One fold acts as the test & others training.

All Thanks To Krish Sir

100 10 10
10 - no 9x 9x 9x
90 - yes 1N 1N 1

(4) Stratified k fold CV

→ It designed for classification problems, where you want the proportion of each class (Eg: Yes & No)

→ For each fold, the ratio in the test & training set remains similar to the original dataset

Eg: 90% Yes 10% No.

100 - datapoint [1 | 1 | 1 | 111]

k = 10.

each k has 9 Yes & 1 No.

(5) Time Series CV

It is used for time-ordered data, such as stock prices.

→ In each fold, you add a bit more data to the training set, making predictions on a slightly later portion of the data.

Data cleaning :

To ~~check~~ clear the error by diff. regions

```
dataset.loc[:122, "Region"] = 0
df.loc[122:, "Region"] = 1
```

fix spaces in colums name.

```
df.columns = df.columns.str.strip()
df.columns
```

Changes the required colums as integer data type:

```
df.columns
Index(['day', 'month', 'year' ------- ])

df[['month', 'day', 'year', temperature. ------ ])
```

Changing the other columns to float data datatype.

```
objects = [features for features in df.columns if df[features].dtypes == 'O']
for i in objects:
    if i != 'classes':
        df[i] = df[i].astype(float)
```

All Thanks To Krish Sir!

EDA & Feature Engineering

Encoding of the categories in classes :

```
df_copy['classes'] = np.where(df_copy['classes'].str.contains
('not fire'), 0, 1)
df_copy.head()
```

Independent & Dependent features

```
X = df.drop('FWI', axis=1)
y = df['FWI']
```

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=0.25, random_state=42)
```

Feature Selection based on correlation :

```
X_train.corr()
```

Check for multicollinearity.

```
plt.figure(figsize=(12,10))
corr = X_train.corr()
sns.heatmap(corr, annot=True)
```

```
def correlation(dataset, threshold):
    col_corr = set()
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i,j]) > threshold:
                colname = corr_matrix.columns[i]
                col_corr.add(colname)
    return col_corr
```

threshold - Domain expertise

```
correlation(X_train, 0.85)
↳ corr_features = correlation(X_train, 0.85)
```

drop features when correlation is more than 0.85

```
X_train.drop(corr_features, axis=1, inplace=True)
X_test.drop(corr_features, axis=1, inplace=True)
```

Feature Scaling or Standardization

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Note : after scaling the data box plot would be easy since all the data will have $\mu = 0$ & $\sigma = 1$.

All Thanks To Krish Sir!

MODELS :

Linear Regression Model :

①
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
linreg = LinearRegression()
linreg.fit(X_train_scaled, y_train)
y_pred = linreg.predict(X_test_scaled)
mae = mean_absolute_error(y_test, y_pred)
score = r2_score(y_test, y_pred)
print("Mean absolute error", mae)
print("R2 score", score)
```

↳ 0.54
↳ 0.98

Lasso Regression	Ridge Regression	Elastic-Net Regression
In line 1: import Lasso	import Ridge	import ElasticNet
lasso = Lasso()	ridge = Ridge()	elastic = ElasticNet
lasso.fit(X_train_scaled, y_train)	ridge.fit(X_train_scaled, y_train)	elastic.fit(X_train_sc, y_train)
y_pred = lasso.predict(X_test_scaled)	y_pred = ridge.predict(X_test_scaled)	y_pred = elastic.pr (X_test_scal
↳ 1.13 ↳ 0.94	↳ 0.56 ↳ 0.984	↳ 1.8 ↳ 0.87

All Thanks To Krish Sir!

Page No.
Date

Cross Validation :

Lasso

```
from sklearn.linear_model
import LassoCV

lassocv = LassoCV(cv=5)

lasscv = fit(X_train
-scaled,
y_train)

lassocv.alpha

↳ 0.057

y_pred = lassocv.predict
(X_train_scaled)
```

Ridge

```
from sklearn.linear_model
import RidgeCV

ridgeCV = RidgeCV(cv=5)

ridgecv.fit(X_train
-scaled,
y_train)

y_pred = ridgecv.predict
(X_test)

ridge cv.alphas

↳ ([0.1, 1, 10])

ridgecv.alpha.

↳ [1]
```

Elastic Net

```
from sklearn.linear_model
import ElasticNetCV

elasticCV = ElasticNetCV
(cv=5)

elasticcv.fit(X_train
_scaled, y_train)

y_pred = elasticcv.predict
(X_test_scaled)

elasticcv.alpha_

↳ 0.043
```

All Thanks To Krish Sir!

2-5.

Pickling

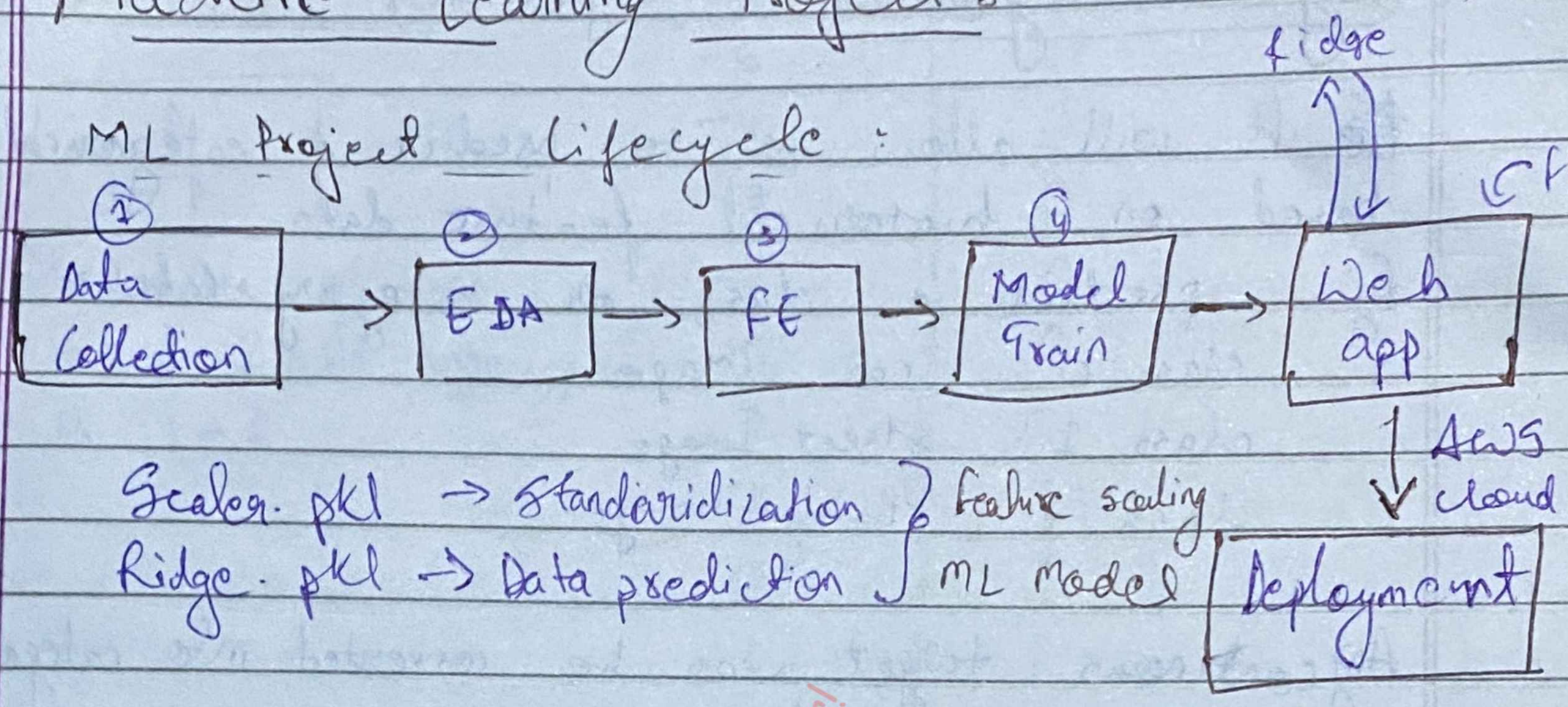
pickle module is used for serialising & de-serialising a python object structure. Any object can be pickled so that it can be saved on disk. What pickle does is that it 'serialises' the object first before writing it to file.

```
import pickle
pickle.dump(regression, open('regressor.pkl', 'wb'))
model = pickle.load(open('regressor.pkl', 'rb'))
model.predict(X_test)

pickle.dump (scaler, open('scaler.pkl', 'wb'))
pickle.dump(ridge, open('ridge.pkl', 'wb'))
pickle
```

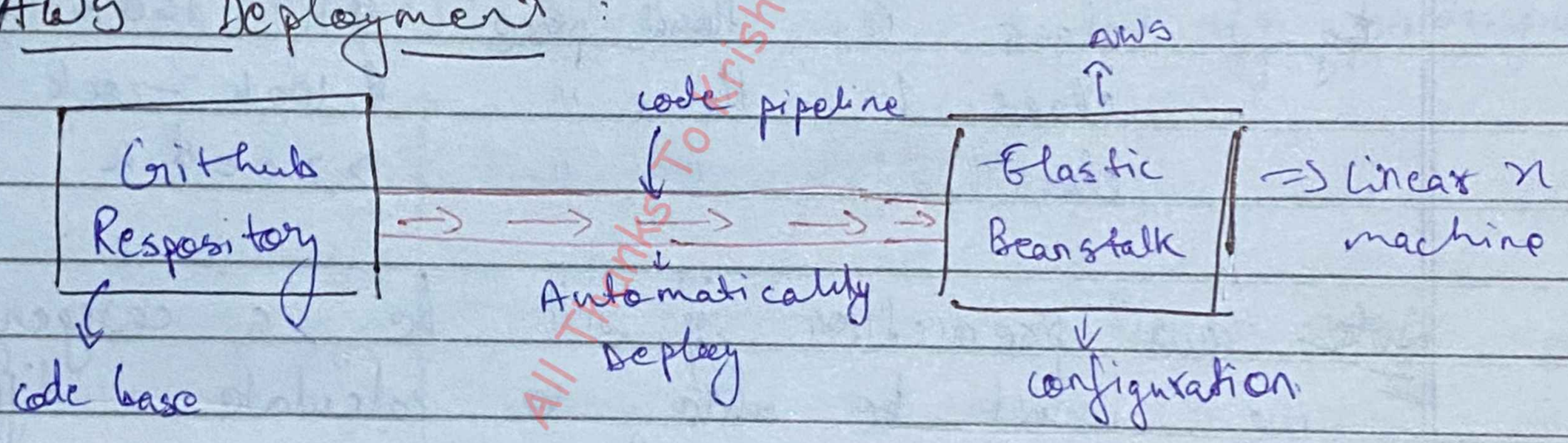
All Thanks To Krishna

Machine Learning Projects :

ML Project Lifecycle :



AWS Deployment :



All Thanks To Krish Sir!

Page No.
Date 1 11 24

Logistic Regression (classification)

It will allow us to predict a categorical label based on historical feature data.

Eg: predict a class or category label:
- class 0 : car image
- class 1 : street Image
- class 2 : Bridge Image.

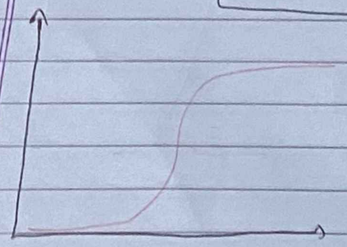
Any continous target can be converted into categories through discretization.

Eg:
- class 0 : House price \$ 0 - 100k
- class 1 : House " \$ 100k - 200k
- class 2 : " " > 200\$ k

Note: our prediction $\hat{y}$ will be a category, meaning we won't be able to calculate a difference basedy on $y - \hat{y}$.

- Logistic Regression works by transforming a linear regression into a classification model through a logistic function.

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



- levelling off behavior
- Transforms any input to be between 0 & 1.

Page No.
Date

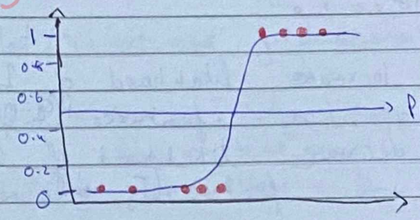
Our data set:

Income	Loan paid
-5	0
-4	0
-3	0
-2	0
-1	0
0	0
2	1
3	1
4	1
5	1

$\boxed{\frac{P}{1-P}}$

The odds of prob. is P & not happening is $1-P$

Treating $P(y=1) >= 0.5$ as a cut-off for classification



$$\hat{y} = \sigma(\beta_0 x_0 + \beta_1 x_1 + \beta_2 x_2 \cdots + \beta_n x_n)$$

$$\hat{y} = \sigma\left(\sum_{i=0}^{n} \beta_i x_i\right)$$

$$\boxed{\hat{y} = \frac{1}{1 + e^{-\sum_{i=0}^{n} \beta_i x_i}}}$$

we can't see any relation betw $\hat{y}$ & β_i

$$\hat{y} + \hat{y} e^{-\sum_{i=0}^{n} \beta_i x_i} = 1$$

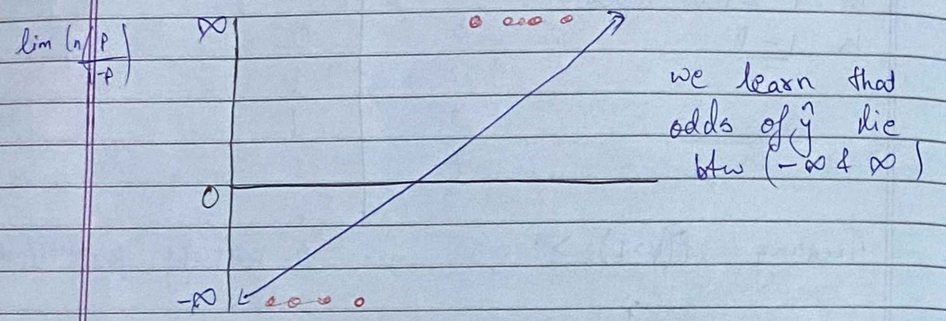
$$\hat{y} e^{-\sum_{i=0}^{n} \beta_i x_i} = 1 - \hat{y}$$

$$\frac{\hat{y}}{1-\hat{y}} = e^{\sum_{i=0}^{n} \beta_i x_i} \quad = (1)$$

All Thanks To Krish Sir

from ① $\ln\left(\frac{\hat{y}}{1-\hat{y}}\right) = \sum_{i=0}^{n} \beta_i x_i$

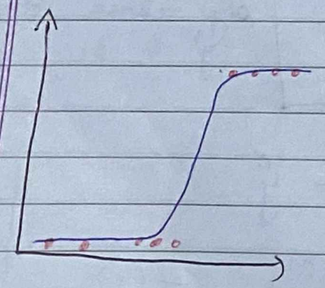
consider $p = 0.5$, halfway point is 0
As p goes to 1 then log odds becomes ∞
As p goes to 0 then log odds becomes $-\infty$



- +ve β increase likelihood of 1 class with increase w.r.to x feature. (Eg: income with shopping)
- -ve β decrease likelihood of 1 class with increase w.r.to x feature. (Eg. incomes with public transport)

How to find best fit line for logistic Regression:

- The first step for max likelihood is to go from log odds back to probability.



$$P = \frac{e^{\ln(\text{odds})}}{1+e^{\ln(\text{odds})}}$$

Logistic Regression → EDA:

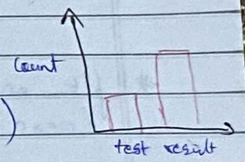
import imp. libraries (like - numpy, pandas, matplotlib)

```
df.describe()
df['test_result'].value_counts()
```

→ ① 3000
② 2000

Visualization:

```
sns.countplot(data=df, x='test_result')
```




```
sns.boxplot(x='test_result', y='age', data=df)
sns.boxplot(x='test_result', y='physical_score', data=df)
sns.scatterplot(x='age', y='physical_score', data=df,
                hue='test_result')
sns.pairplot(df, hue='test_result')
sns.heatmap(df.corr(), annot=True)
```

3-D scatterplot:

```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(df['age'], df['physical_score'], df['test_result']
           c=df['test_result'])
```

~~# The scatter~~

All Thanks To Krish Sir!

Fit & Transform :

fit : we are asking to learn from the data
The scaler calculates mean & std. dev.

we are training the scaler to know the data ; But it doesn't actually change data

Transform : Once the scaler knows the mean & std. dev, we can transform the data. Eg : z-score.

fit on X-train to learn about the training data's distribution (mean, std)

transform on both X-train & X_test (to apply that learned scaler)

Independent Feature & Dependent feature :

```
X = df.drop['test_result']
y = df['test_result']
```

Train test split

```
from ~~import~~ sklearn.preprocessing import ~~train_test_split~~ standardScaler
from      "   .model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1
                                                    random_state = 101)
Scaler = StandardScaler()
Scaled_X_train = scaler.fit_transform(X_train)
Scaled_X_test = scaler.transform(X_test)
```

Model Training :

```
from sklearn.linear_model import LogisticRegression
# help (LogisticRegression)

log_model = LogisticRegression()
log_model.fit(Scaled_X_train, y_train)

log_model.coef_
↳ array([[-0.9449 , 3.4549]])

y_pred = log_model.predict_proba(Scaled_X_test)
```

Performance Metrics :

Confusion metrics :

4 possible outcomes
- Infected person test positive
- Healthy person test negative
- Infected person test -ve
- Healthy person test +ve

Prediction	Actual: Infected	Actual: Healthy
Infected	(True +ve)	(False +ve)
Healthy	(False -ve)	(True -ve)

Suppose 5% Inf, 95% Healthy

	Inf.	Health
Inf	4	2
Healthy	1	93

Q. What is accuracy ?
How often is the model correct ?

$$Acc = \frac{(TP + TN)}{Total}$$

$$Acc = \frac{4 + 93}{100}$$

= 97% Accuracy.

All Thanks To Krishna Sir

Page No.
Date

Accuracy paradox:

	Inf	Healthy
Inf	0	0
Healthy	5	95

Acc = 95%

Imbalanced = Where ratio are different

Imbalanced classes will always result in a distorted accuracy

In the above example we predicted everyone is healthy which was wrong but still got 95% acc so. we cant rely solely on accuracy as a metric. & try others (precision, recall & etc)

Metrics to use to handle accuracy paradox.

1. recall
2. precision
3. F-1 score

Recall: When it actually is a +ve case, how often is it correct

$$\text{Recall} = \frac{TP \text{ (true +ve)}}{\text{Total Actual Positives}}$$

In previous example $\frac{4}{5} = 0.8$

Precision: When prediction is +ve, how often is it correct?

$$\text{precision} = \frac{TP \text{ (true +ve)}}{\text{Total Predicted +ves}}$$

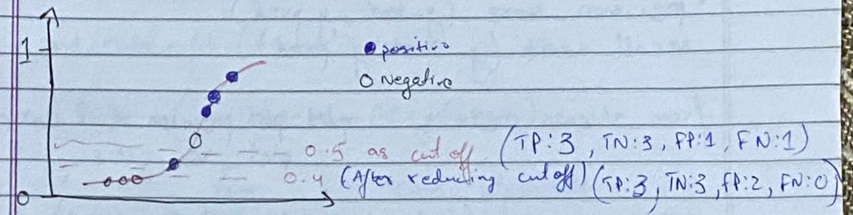
$\frac{2}{3} = 0.666$

F-1 score: harmonic mean of precision & recall.

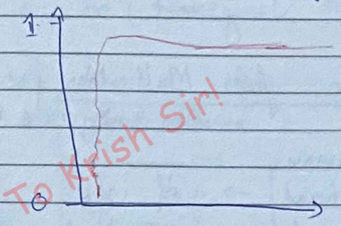
$$F = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

Page No. 9
Date 4 11 24

ROC curves (Receiver Operator characteristic).



AUC → Area Under the curve, allows us to compare ROCs for different models



All Thanks To Krish Sir!

Continue from model training.

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

y_pred = log_model.predict(scaled_X_test)

accuracy_score(y_test, y_pred)
↳ 0.93

confusion_matrix(y_test, y_pred)
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(log_model, scaled_X_test, y_test)
print(classification_report(y_test, y_pred))
```

```
from sklearn.metrics import precision_score, recall_score
precision_score (y_test, y_pred)
recall_score (y_test, y_pred)

from sklearn.metrics import plot_precision_recall_curve,
                            plot_roc_curve

plot_roc_curve (log_model, scaled_X_test, y_test)

plot_precision_recall_curve (log_model, scaled_X_test,
                              y_test)
```

(Online)

Logistic Regression for Multiclass problems:

(Non-Binary: more than 2 Answers)

```
import all imp libraries
df = pd.read → df.head → df.info
After creating dependent & independent features
& scaling them

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

log_model = LogisticRegression (solver = 'saga', multi_class = 'ovr',
                                max_iter = 5000)

penalty = ['l1', 'l2', 'elasticnet']
l1_ratios = np.linspace (0, 1, 20)
C = np.logspace (0, 10, 20)
param_grid = { 'penalty' : penalty,
               'l1_ratio' : l1_ratio,
               'C' : C }
```



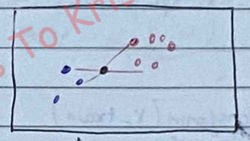
```
grid_model = GridSearchCV (log_model, param_grid = param_grid)

grid_model.fit (scaled_X_train, y_train)
```

KNN (K nearest neighbour)

→ Its diff. from K-means algorithm
→ used for classification & not for regression

It simply assigns a label to new data based on the distance the old data & new data.

Eg: To train an algorithm to identify the sex of a new baby chick based on historical features



K = 4 leads to a tie. 1) Always choose odd K.
2) simply reduce K by 1.
3) Randomly break tie

What does Scikit-learn do in case of tie?

- The answer will be the class that happens to appear first in the set of neighbors.
- Results are ordered by distance, so it chooses the class of the closest point.

We want a K-value that minimizes error.

- Error = 1 - Accuracy

2 methods : Elbow method

All Thanks To Krish Sir!!

KNN Algorithm :

- choose K value ↗ No of features.
- Sort feature vectors (N-dimension space) by distance metric.
- choose class based on K

Distance Metric : (scaling is necessary for KNN)

- Euclidean
- Manhattan
- Minkowski

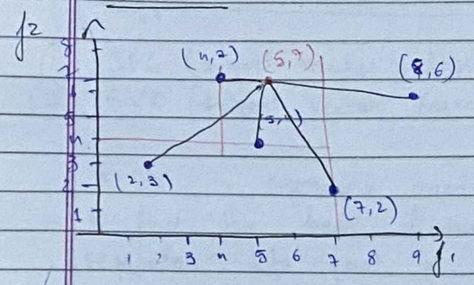
Train test split :

```
X = df.drop('Cancer Present', axis=1)
y = df['Cancer Present']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)

scaler = StandardScaler()
scaled_X_train = scaler.fit_transform(X_train)
scaled_X_test = scaler.transform(X_test)
from sklearn.neighbors import KNeighborsClassifier

knn_model = KNeighborsClassifier(n_neighbors = 1)
knn_model.fit(scaled_X_train, y_train)
```

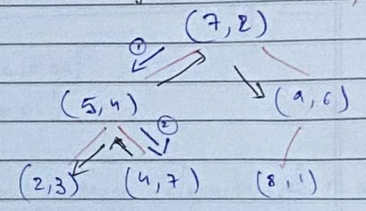
Model Evaluation

```
y_pred = knn_model.predict(scaled_X_test)
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score.
```

① KD Tree

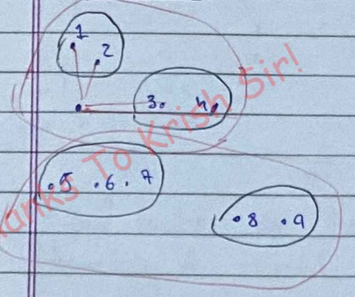


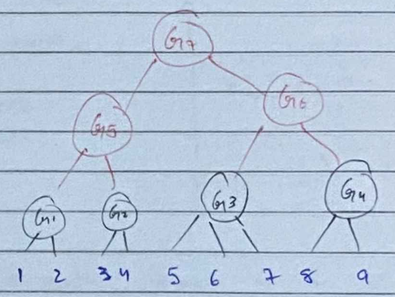
Binary Tree (KD tree)



Back tracking :

② Ball Tree





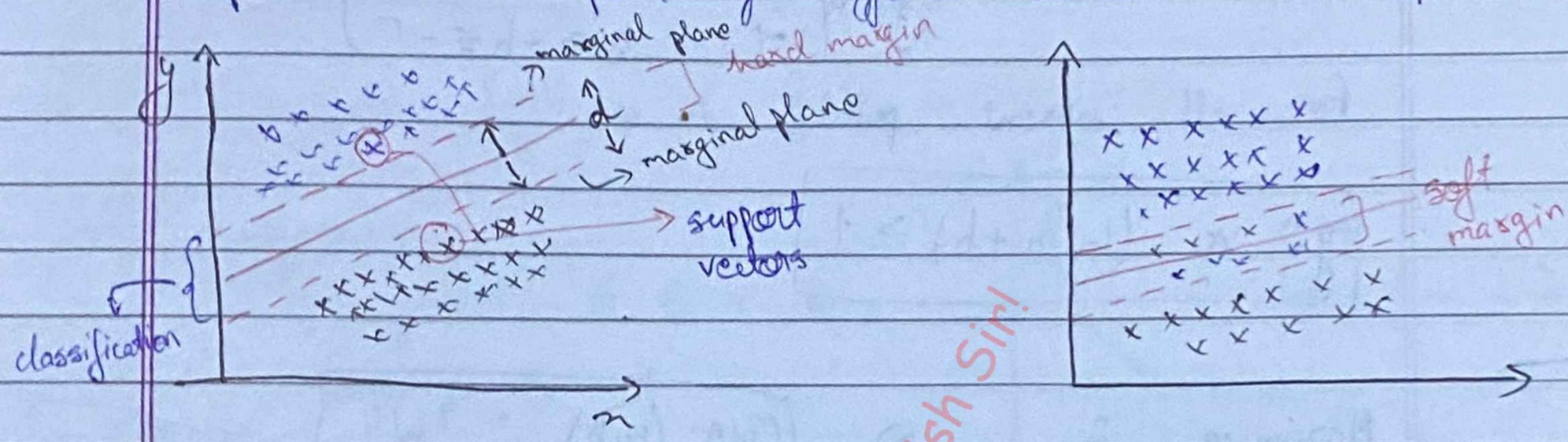
All Thanks To Krish Sir!

Page No.
Date 8 11 24

Support Vector Machines ML algorithm

1. SVC (support vector classification)
2. SVR (support vector Regression)

It is a supervised machine learning algorithm & it aims to find the best decision boundary or hyperplane, that separates data point of different classes with max. possible



Support vectors are the data points closest to the decision boundary & crucial for making decision boundary.

Marginal planes are parallel to the optimal hyperplane. The distance btw. marginal plane is to optimized

Maths Intuition :

$ax + by + c = 0$

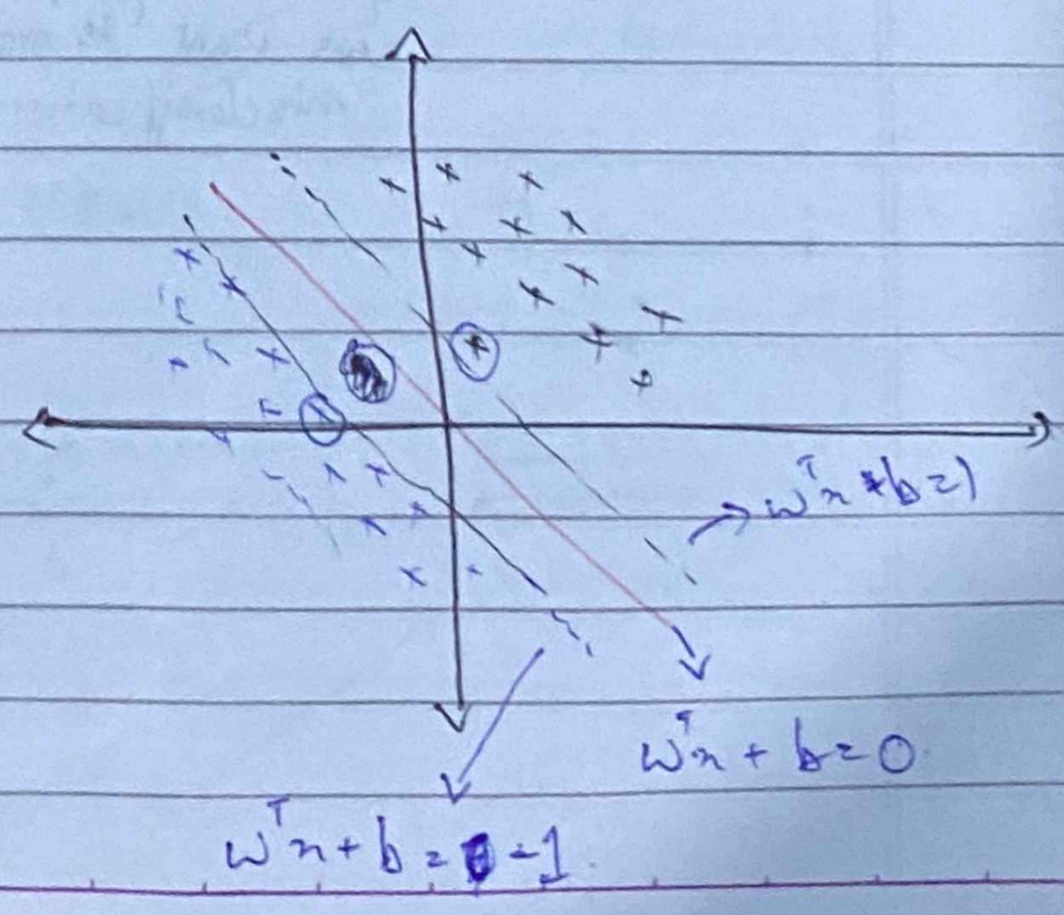
$w_1 x_1 + w_2 x_2 + b = 0$

$b = 0$

$$\boxed{w^T x = 0}$$

$d = -ve$ (below plane)

$d = +ve$ (above plane)



All Thanks To Krish Sir!

$$w^T x_1 + b = 1$$
$$w^T x_2 + b = -1$$
$$w^T(x_1 - x_2) = +2$$

$$\frac{w^T(x_1 - x_2)}{\|w\|} = \frac{+2}{\|w\|}$$

Cost function : Max. $\frac{2}{\|w\|}$

Constraint such that

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

For all current points

$$\boxed{y_i \times (w^T x + b) \geq 1}$$

$$\underset{w,b}{\text{Maximize}} \ \frac{2}{\|w\|} \Rightarrow \boxed{\text{Min.}(w,b) = \frac{\|w\|}{2}}$$

Cost function of SVM (svc) :

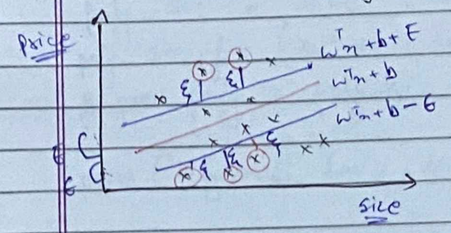
$$\text{Min}(w,b) \ \frac{\|w\|}{2} + \boxed{C_i \sum_{i=1}^{n} \xi_i} \Rightarrow \text{Hinge Loss.}$$

C_i → hyperparameter to accept errors.

C_i → {How many points we want to avoid misclassification}

$\sum \xi_i$ → Summation of the dist. of the incorrect data points from the marginal plane.

Support Vector Regression (SVR)



ϵ → Marginal Error

ξ → points outside are penalized but not the inside ones

Cost function :

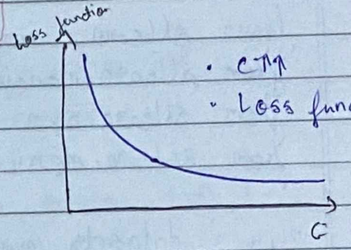
$$\text{Min.}(w,b) \ \frac{\|w\|}{2} + \boxed{C \sum_{i=1}^{n} \xi_i} \rightarrow \text{Hinge Loss.}$$

C → Hyperparameter

Constraint :

$$|y_i - w_i x_i| \leq \epsilon + \xi_i$$

$|y_i - w_i x_i|$ → Loss function

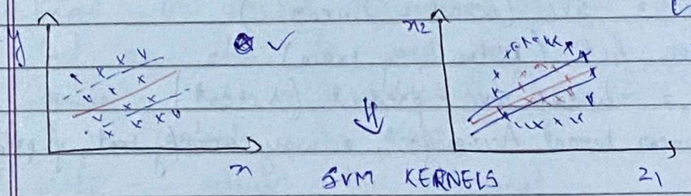


- $C \uparrow$
- Loss func $\downarrow\downarrow$

C : controls the regularization.

Unlike in SVC in SVR we create a best fit line which has certain tolerance acc. to datapoints.

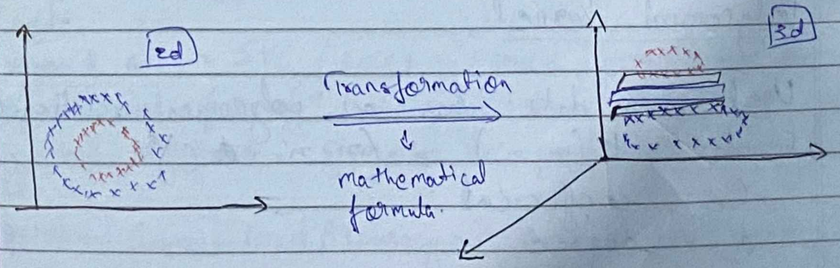
SVM KERNELS : transforms into higher dimensional



Linear SVC ✗

Acc $\downarrow\downarrow$

SVM KERNELS



All Thanks To Krish Sir

Types of svm kernel:

1. Polynomial Kernel
2. RBF Kernel
3. Sigmoid Kernel

1. Linear Kernel

Works best when data is linearly seperable.

Code:

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

x, y = datasets.make_classification(n_samples=100, n_features=2,
                                    random_state = 1)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size
                                    = 0.3, random_state = 1)
```

* SVM with linear kernel

```
linear_svc = SVC (kernel = 'linear')
linear_svc.fit (x_train, y_train)
y_pred = linear_svc.predict (x_test)
print ("Linear kernel Accuracy:", accuracy_score(y_test, y_pred))
```

2. Polynomial Kernel

Used when data has a polynomial relationship.

Formula: $K(x, x') = (x \cdot x' + c)^d$

c = coefficient

d = degree

Code:

```
poly_svc = SVC (kernel = 'poly', degree = 3, coef0 = 1)
poly_svc.fit (x_train, y_train)
y_pred_poly = poly_svc.predict (x_test)

print ("Polynomial kernel Accuracy:", accuracy_score(y_test, y_pred_poly))
```

3. Radial Basis Function (RBF) Kernel

Popular for non-linear data, as it maps samples into an infinite-dimensional space.

Formula: $K(x, x') = \exp(-\gamma |x - x'|^2)$

γ = controls the influence

Code:

```
rbf_svc = SVC (kernel = 'rbf', gamma = 'scale')
rbf_svc.fit (x_train, y_train)
y_pred_rbf = rbf_svc.predict (x_test)
print ("RBF kernel Accuracy:", accuracy_score(y_test, y_pred_rbf))
```

4. Sigmoid Kernel

Used in neural networks as an activation function but can also be used for svm

Formula: $K(x, x') = \tanh(\gamma \cdot x \cdot x' + c)$

γ – controls the slope

coef0 – adjust the intercept

Code:

```
sigmoid_svc = SVC (kernel = 'sigmoid', gamma = 'scale', coef0 = 0.5)
sigmoid_svc.fit (x_train, y_train)
y_pred_sigmoid = sigmoid_svc.predict (x_test)

print ("Sigmoid kernel Accuracy:", Accuracy_score (y_test, y_pred_
sigmoid))
```

All Thanks To Krish Sir!

SVR Implementation :

Feature Encoding & One hot Encoder

```
from sklearn.preprocessing import LabelEncoder

le1 = LabelEncoder()
le2 = LabelEncoder()
le3 = LabelEncoder()
X_train['sex'] = le1.fit_transform(X_train['sex'])
X_train['smoker'] = le2.fit_transform(X_train['smoker'])
X_train['time'] = le3.fit_transform(X_train['time'])
X_test['sex'] = le1.transform(X_test['sex'])
X_test['smoker'] = le2.transform(X_test['smoker'])
X_test['time'] = le3.transform(X_test['time'])

from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder

ct = ColumnTransformer(transformers = [('onehot', OneHotEncoder
                                        (drop='first'), [3]], remainder = 'passthrough')

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

SVR

```
from sklearn.svm import SVR
svr = SVR()
svr.fit(X_train, y_train)
y_pred = svr.predict(X_test)
```

Hyperparameter Tuning using GridSearchCV

```
from sklearn.model_selection import GridSearchCV

param_grid = {'C' : [0.1, 1, 10, 100, 1000],
              'gamma' : [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel' : ['rbf']}

grid = GridSearchCV(SVR(), param_grid, refit = True, verbose=3)
grid.fit(X_train, y_train)
grid.best_params_
grid_prediction = grid.predict(X_test)
```

All Thanks To Krish Sir!

Page No.
Date 13 11 24

Naive Bayes Algorithm (classification)

Conditional Prob. : when a certain event has already occurred.

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

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

$P(A/B)$ = Prob of Event A given B has occurred

I/P features			dependent
x_1	x_2	x_3	y
–	–	–	Yes
–	–	–	No
–	–	–	Yes
–	–	–	No

$$P(y/x_1, x_2, x_3) = \frac{P(y) \times P(x_1, x_2, x_3 / y)}{P(x_1, x_2, x_3)}$$

$$\boxed{P(y/x_1, x_2, x_3) = \frac{P(y) \times P(x_1/y) \times P(x_2/y) \times P(x_3/y)}{P(x_1) \times P(x_2) \times P(x_3)}}$$

$$\Downarrow$$

$$P(Yes/x_1, x_2, x_3) = \frac{P(yes) \times P(x_1/yes) \times P(x_2/yes) \times P(x_3/yes)}{\underbrace{P(x_1) \times P(x_2) \times P(x_3)}_{const}}$$

$$P(No/x_1, x_2, x_3) = \frac{P(No) \times P(x_1/No) \times P(x_2/No) \times P(x_3/No)}{\underbrace{P(x_1) \times P(x_2) \times P(x_3)}_{const}}$$

Nikalega dono, jo zyada hoga usse le lega.

Page No.
Date

Day Outlook Temp. Humidity Wind Play Tennis : } Dataset.

Outlook:

	Yes	No	P(E/Yes)	P(E/No)
Sunny	2	3	2/9	3/5
Overcast	4	0	4/9	0
Rain	3	2	3/9	2/5

Temp:

Hot	2	2	2/9	2/5
mild	4	2	4/9	2/5
cold	3	1	3/9	1/5

Play Tennis (Y/N)

Yes – 9 P(yes) = 9/14

No – 5 P(No) = 5/14.

$$P(Yes/sunny, Hot) = P(Yes) \times P(sunny/yes) \times P(Hot/yes)$$

$$= \frac{9}{14} \times \frac{2}{9} \times \frac{2}{9} = \frac{2}{63} = 0.03$$

$$P(No/sunny, Hot) = P(No) \times P(sunny/No) \times P(Hot/No)$$

$$= \frac{5}{14} \times \frac{3}{5} \times \frac{2}{5} = \frac{3}{35} = 0.085$$

$$P(Yes/sunny, hot) = \frac{0.031}{0.031 + 0.085} = 0.27 = 27\%$$

$$P(No/sunny, hot) = \frac{0.085}{0.031 + 0.085} = 0.73 = 73\%$$

Output = person is not going to play tennis.

All Thanks To Krish Sir!

Variants of Naive Bayes:

① Bernoulli Naive Bayes:

Whenever your features are following a Bernoulli Distribution, then we need to use Bernoulli Naive Bayes Algorithm.

Dataset:

f_1	f_2	f_3	O/P
Yes	Pass	Male	Yes
Yes	Fail	female	No
No	Pass	male	Yes
Yes	fail	female	No.

② Multinomial Naive Bayes:

Dataset : Spam classification

Email Body	Spam / Not spam
• You have won million $$	Spam
• Krish You have done good job.	Not Spam

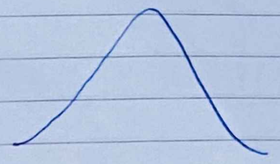
⇓

Numerical Values ⇒ NLP (Natural Language Processing)

1. BOW
2. Tf - Idf
3. Word 2 vec

③ Gaussian naive Bayes:

If the features are following Gaussian Distribution, then we use Gaussian Naive Baye's



Dataset → Continous

↳ age Height Weight

Practical implementation:

```
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()
gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)
```

Here we have numerical value that why we used GaussianNB

All Thanks To Krish Sir!

Decision Tree classifier:

→ ID3

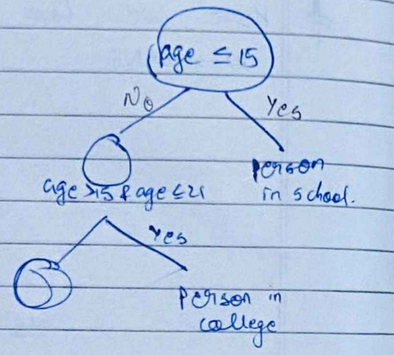
→ CART :- (mostly used)

a) Entropy and Gini Index → Purity split

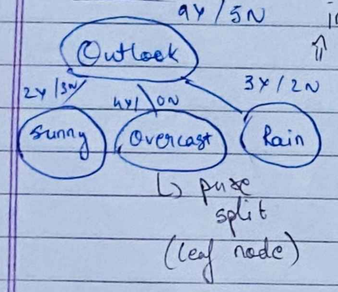
b) Information gain → features to select for DT construction.

Eg: age = 14

```
if (age ≤ 15):
    Print ("The person is in school)
elif (age > 15 and age ≤ 21):
    Print ("Person maybe in college")
else
    Print ("Person has passed")
```



Dataset:



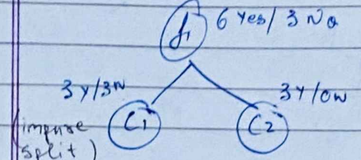
① Purity → Pure or Impure split
   a) Entropy
   b) Gini impurity

② What feature you need to select for leaf node

Entropy

$$H(S) = -P_+ \log_2 P_+ - P_- \log_2 P_-$$

(P_+: prob. yes, P_-: prob. of no)



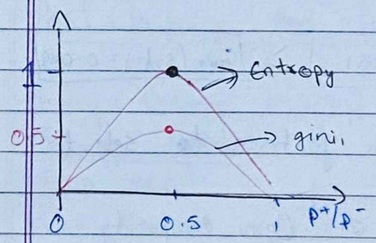
$$H(C_1) = -P_+ \log P_+ - P_- \log_2 P_-$$

$$= -\frac{3}{6}\log_2\frac{3}{6} - \frac{3}{6}\log_2\frac{3}{6}$$

$$= 1 \text{ (Impure split)}$$

$$H(C_2) = -\frac{3}{3}\log_2\frac{3}{3} - \frac{3}{3}\log_2\frac{3}{3}$$

$$= 0$$



Entropy ranges btw. 0 & 1.

Gini Impurity:

$$G.I = 1 - \sum_{i=1}^{n} (P)^2$$

$$= 1 - ((P_+)^2 + (P_-)^2)$$

$$= 1 - \left(\left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^2\right)$$

$$= 0.5$$

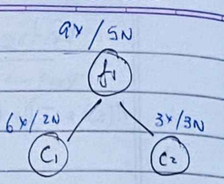
Gini ranges btw. 0 & 0.5

All Thanks To Krish Sir!

Page No.
Date 15 11 24

Information Gain :

$$Gain(S, f_1) = \underbrace{H(S)}_{\text{Entropy of root node}} - \sum_{v \in val} \frac{|S_v|}{|S|} \underbrace{H(S_v)}_{\text{sum of values of } C_1 \& C_2}$$



$$H(S) = -P_+ \log_2 P_+ - P_- \log_2 P_-$$

$$= -\frac{9}{14} \log_2 \frac{9}{14} - \frac{5}{14} \log_2 \frac{5}{14}$$

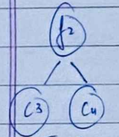
$$\approx 0.94$$

$$H(C_1) = -\frac{6}{8} \log_2 \frac{6}{8} - \frac{2}{8} \log_2 \frac{2}{8}$$

$$\boxed{H(C_1) = 0.81} \quad \boxed{H(C_2) = 1}$$

$$Gain(S, f_1) = 0.94 - \left[\frac{8}{14} \times 0.81 + \frac{6}{14} \times 1\right]$$

$$\boxed{Gain(S, f_1) \approx 0.049}$$



$$\boxed{Gain(S, f_2) = 0.051} > \boxed{Gain(S, f_1) = 0.049}$$

I.G helps to find which features to select

When to use entropy or Gini impurity.

O/P = 3 categories

$$H(S) = -P_{C_1} \log_2 P_{C_1} - P_{C_2} \log_2 P_{C_2} - P_{C_3} \log_2 P_{C_3}$$

Whenever dataset is small → Entropy (10000 record)

" dataset is large → Gini Impurity

By default Gini is used.

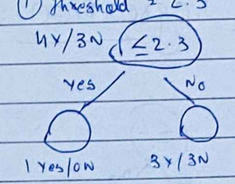
Page No.
Date

Decision tree split for Numerical features :

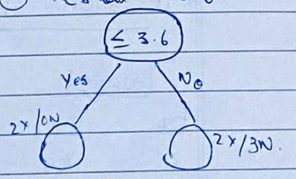
f_1	O/P
2.3	Yes
3.6	Yes
4	No
5.2	No
6.7	Yes
8.9	No
10.5	Yes

① sort the feature value

① Threshold = 2.3



② threshold = 3.6



we will find Infor. Gain for every threshold then the threshold with highest I.G is selected as root node.

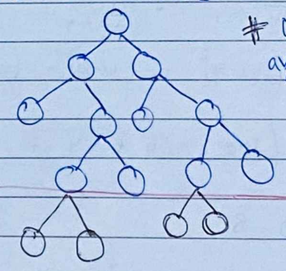
Dis advantage :
Not suitable for very big records like million datapoints due to time complexity.

All Thanks To Krish Sir!

Post Pruning And Pre Pruning (Decision Tree)

Pruning : Removing unwanted branches

Training dataset
Train ACC ↑
Test Acc ↓
Overfitting



We prune DT to avoid overfitting

Post Pruning : 1) Construct the DT
2) Prune it with respect to depth
3) Smaller datasets.

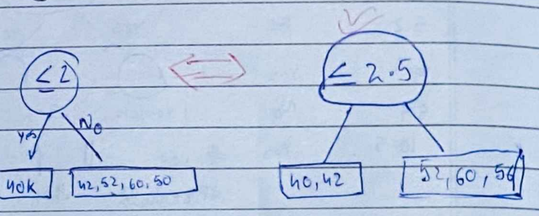
Pre Pruning : 1) Hyperparameter Tunning
2) While contructing DT.

Hyperparameter tuning: It's the process to find optimal values for these parameters to achieve the best model performance

Decision Tree Regression: (Continous feature)

Dataset

Exp	Gap	Salary
2	Yes	40k
2.5	Yes	42k
3	No	52k
4	No	60k
4.5	Yes	56k



We can not use entropy, Gini & Information gain so we use Variance Reduction.

Variance Reduction (Regression Problem)

$$\text{Variance} = \frac{1}{n}\sum_{i=1}^{n}(y-\bar{y})^2 \qquad \bar{y} = 50k$$

($\bar{y}$ → Avg)

$$= \frac{1}{5}\left((40-50)^2 + (42-50)^2 + (52-50)^2 + (60-50)^2 + (56-50)^2\right)$$

$$= \frac{1}{5}\left(100 + 64 + 4 + 100 + 36\right)$$

$$= 60.8$$

$$V(c_1) = \frac{1}{n}\sum_{i=1}^{n}(y-\bar{y})^2 = \frac{1}{1}(40-50)^2 = 100$$

$$V(c_2) = \frac{1}{n}\sum_{i=1}^{n}(y-\bar{y})^2 = \frac{1}{4}\left[(42-50)^2 + (52-50)^2 + (60-50)^2 + (56-50)^2\right]$$

$$= \frac{1}{4}\left[64 + 4 + 100 + 36\right] = 51$$

ratio

Variance Reduction: $= Var(\text{Root}) - \sum w_i \, Var(\text{child})$

$$= 60.8 - \left[\frac{1}{5}\times 100 + \frac{4}{5}\times 51\right]$$

$$= 60.8 - 20 - 40.8$$

$$= 0$$

$$V.R(f_1) = 0$$

Variance of f_2: 60.8

$$V(c_1) = \frac{1}{n}\sum_{i=1}^{n}(y-\bar{y})^2 = \frac{1}{2}\left[(40-50)^2 + (42-50)^2\right] = \frac{1}{2}\left[100 + 64\right]$$

$$= \frac{164}{2} = 82$$

$$V(c_2) = \frac{1}{3}\left[(52-50)^2 + (60-50)^2 + (56-50)^2\right] = \frac{1}{3}\left[4 + 100 + 36\right]$$

$$= \frac{140}{3} = 46.66$$

$$V.R(f_2) = 60.8 - \left[\frac{2}{5}\times 82 + \frac{3}{5}\times 46.66\right]$$

$$= 60.8 - 60.76$$

$$= 0.04$$

$V.R$ (left split) $<$ $V.R$ (Right split)

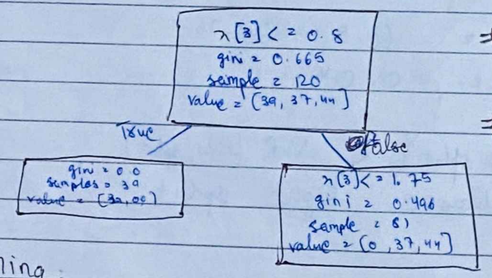
so we choose Right split.

All Thanks To Krish Sir!

Decision Tree classifier → Implementation :

```
from sklearn.tree import DecisionTreeClassifier
tree classifier = DecisionTree classifier()
tree classifier.fit (x_train, y_train)
```

```
## Visualize the Decision Tree
from sklearn import tree
plt.figure (figsize = (15,10))
tree.plot_tree (tree classifier, filled = True)

y_pred = tree classifier.predict (x_test)

from sklearn.metrics import confusion_matrix, classification_report

cm = confusion_matrix (y_test, y_pred)
print (cm)
print (classification_report (y_test, y_pred))
```



Left is always True

Right is always False.

Pruning :

```
Param = { 'criterion' : ['gini', 'entropy', 'log_loss'],
          'splitter' : ['best', 'random'],
          max_depth : [1, 2, 3, 4, 5],
          max_features : ['auto', 'sqrt', 'log2'] }
```

```
from sklearn.model_selection import GridSearchCV
treemodel = DecisionTree Classifier ()
grid = GridSearchCV (treeclassifier, param_grid = param, cv = 5,
                     scoring = 'accuracy')
grid.best_params_
```

Decision Tree Regressor - Implementation :

```
from sklearn.tree import DecisionTreeRegressor
regressor = Decision Tree Regressor () → regressor.fit (X_train, y_train)
grid = GridSearchCV (regressor, param_grid = param, cv = 5, scoring = 'neg_mean_
                                                                squared_error')
grid.fit (x_train, y_train)
grid.best_params
```

```
# Put best params in selectedmodel

selectedmodel = Decision Tree Regressor (criterion = 'friedman_mse', max_depth = 4,
                                         max_features = 2)
selected model.fit (x_train, y_train)

y_pred = selected model.predict (x_test)
```

All Thanks To Krish Sir!

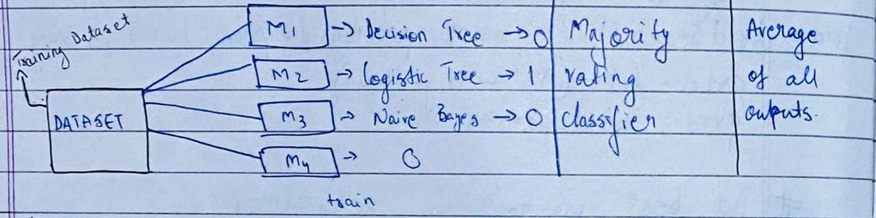
Page No.
Date 16 11 24

Ensemble Techniques : Bagging & Boosting

They helps us to combine multiple algorithms together.

Bagging : ① Random Forest Algorithm (Base learner)

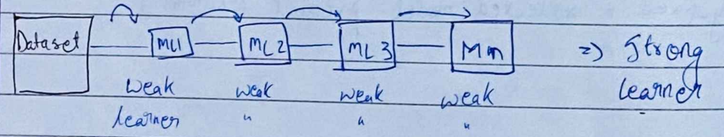
Idea Behind bagging :



All the base learners parallely is bagging

Boosting : ① AdaBoost ② Gradient Boost

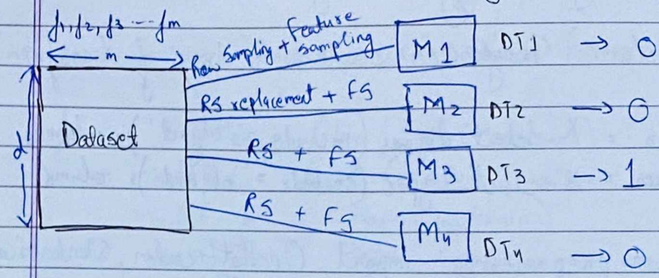
Idea behind boosting :



Boosting is an iterative process where every weak learner is trained to correct the errors of its predecessor, gradually enhancing the overall models performance.

Page No.
Date

Random Forest : (Classification & Regression)



In replacement majority of rows & features will replace but some will be the same.

Classification - majority voting classifier
Regression - Avg of output

Q. Why should we use Random Forest instead of DT ?

① less overfitting (By avg / voting which makes it robust to variation)
② Improved Accuracy (due to multiple training models)
③ Handling high variance
④ Stability to Data fluctuation.

Overfitting Train Acc ↑ (Low Bias) → Random forest.
Test Acc ↓ (High Variance) → Low Variance

All Thanks To Krish Sir!

8/11/24

Page No.
Date

Feature Engineering :

create Column Transformer with 3 types of transformers

```
cat_features = X.select_dtypes(include = 'object').columns
num_features = X.select_dtypes(exclude = 'object').colums

from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer

numeric_transformer = StandardScaler()
oh_transformer = OneHotEncoder(drop = 'first')

preprocessor = ColumnTransformer(
        [
            ("OneHotEncoder", oh_transformer, cat_features),
            ("StandardScaler", numeric_transformer, num_features)
        ] )
X_train = preprocessor.fit_transform(X_train)
```

Now X_train will be converted into all numeric data.

Applying Random Forest :

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score,
                            classification_report, confusion_matrix,
                            precision_score, recall_score, f1_score,
                            roc_curve, roc_auc_score
```

Page No.
Date

1st time training an efficient model

```
models = { "Random Forest" : RandomForestClassifier() }

for i in range(len(list(models))) :
    model = list(models.values())[i]
    model.fit(X_train, y_train)
```

make predictions

```
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)
```

Training set performance

```
model_train_accuracy = accuracy_score(y_train, y_train_pred)
model_train_f1 = f1_score(y_train, y_train_pred, average='weighted')
model_test_accuracy = accuracy_score(y_test, y_test_pred)
model_test_f1 = f1_score(y_test, y_test_pred, average='weighted')
```

In this efficient model the ml model is in form of dictionary which iterate in for loop so you can add multiple ML models in form of key-value pairs to get prediction from diff. models

```
print(list(models.keys())[i])
print('Model performance for Training set')
print("Accuracy : {:.4f}".format(model_train_accuracy))
print("Recall : {:.4f}".format(model_train_recall))
"                    " .format(model_test_accuracy))
"                    " .format(model_train_recall))
```

O/P → Train data
Acc – 1
Recall – 1

Test data
Acc – 0.93
Recall – 0.66.

All Thanks To Krish Sir

Hyperparameter Training

```
rf_params = {"max_depth": [5, 8, 15, 0, 10],
             "max_features": [5, 7, "auto", 8],
             "min_samples_split": [2, 8, 15, 20],
             "n_estimators": [100, 200, 500, 1000]}
```

Efficient way

Models list for Hyperparameter tuning

```
randomCV_models = [("RF", RandomForestClassifier(), rf_param),]
```

Performing Hyperparameter tuning

```
from sklearn.model_selection import RandomizedSearchCV
model_param = {}
for name, model, params in randomcv_models:
    random = RandomizedSearchCV(estimator=model,
                                param_distribution=params
                                n_iter=100,
                                cv=3,
                                verbose=2,
                                n_jobs=-1)
random.fit(X_train, y_train)
model_param[name] = random.best_params_

for model_name in model_param:
print(f"---------- Best Params for {model_name}
print(model_param[model_name])
```

O/P ↳ ---------- Best Params for RF ----------
'n_estimator': 1000, 'min_samples_split': 2, 'max_features': 7,
'max_depth': None }

Put Best params in model
eg. RandomForestClassifier(----------)

All Thanks To Krish Sir!

Data cleaning :

Handling Missing values
- Handling missing values
- Handling Duplicates
- check data type
- Understand the data.

Random Forest Regression :

import all Libraries

create a function to Evaluate Model

```
def evaluate_model(true, predicted):
    mae = mean_absolute_error(true, predicted)
    mse = mean_squared_error(true, predicted)
    rmse = np.sqrt(mean_squared_error(true, predicted))
    r2_square = r2_score(true, predicted)
    return mae, rmse, r2_square
```

Beginning Model Training

```
models = {"Linear Regression": LinearRegression(),
          "Lasso": Lasso(),
          "Ridge": Ridge(),
          "K-Neighbors Regressor": KNeighborsRegressor(),
          "Decision Tree": DecisionTreeRegressor()
          "RandomForest": RandomForestRegressor(), }

for i in range(len(list(models))):
    model = list(models.values())[i]
    model.fit(X_train, y_train)
```

Page No.
Date

```
## Make predictions
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

## Evaluate train & test dataset

model_train_mae, model_train_rmse, model_train_r2 = evaluate_model
                                                    (y_train, y_train_pred)
model_test_mae, model_test_rmse, model_test_r2 = evaluate_model
                                                 (y_test, y_test_pred)

print (list (models.keys()) [i])
print ("ms E : {:.4f}".format (model_train_rmse))
print ("R2_score : {:.4f}".format (model_train_r2))
```

All Thanks To Krish Sir!

ADA BOOST Machine Learning Algorithms :

Assigns weight to the weak learner.

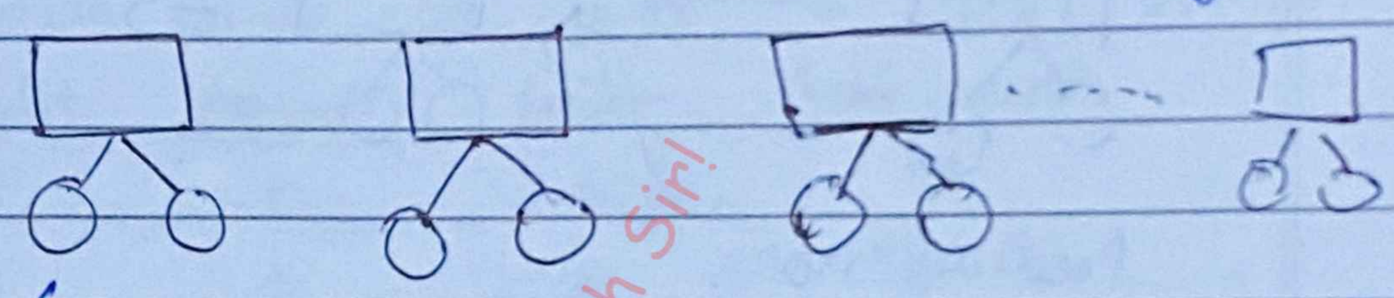
$$\boxed{f = \alpha_1(m_1) + \alpha_2(m_2) + \alpha_3(m_3) + \cdots \alpha_n(m_n)}$$

$m_1 \cdots m_n \rightarrow$ Decision Tree stumps

$\alpha_1, \alpha_2, \alpha_3 \cdots \alpha_n \Rightarrow$ weights

↳ Classification
↳ Regression

Decision stump :



Depth = 1 :

↳ weak learner

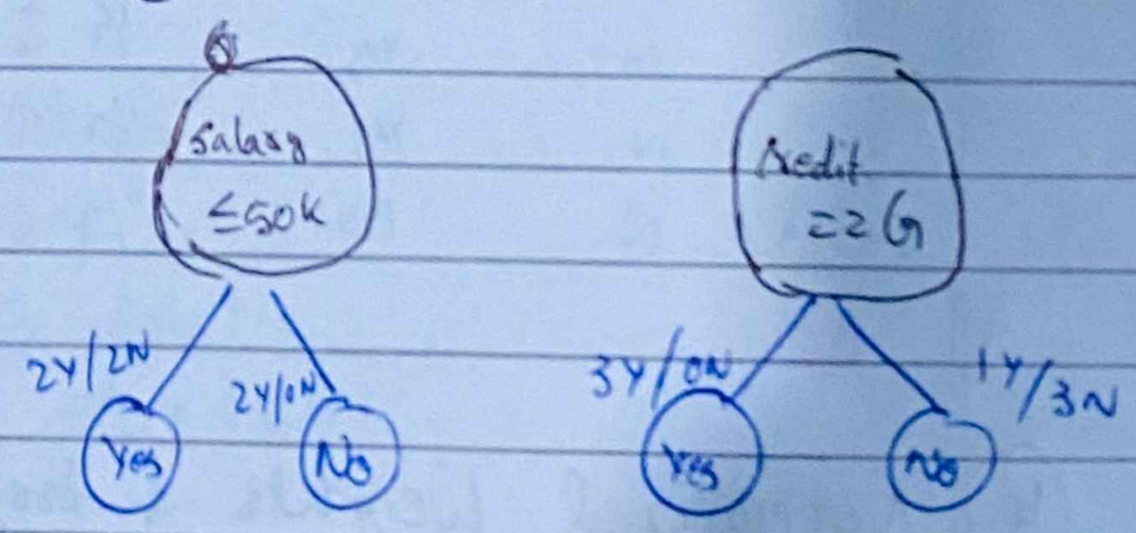
Underfitting
↳ Train Acc ↓ 40%
↳ Test Acc ↑ 45%

{ High Bias, Low Variance } ⇒ { Low Bias, Low variance }

Maths Intuition :

Salary	Credit	Approach
<=50k	B	NO
<=50k	G	Yes
<=50k	G	Yes
>50k	B	No
>50k	G	Yes
>50k	N	Yes
<=50k	N	NO

① we create Decision tree stump



we select the best DT stump using Entropy or G.I

② Sum of Total Errors & Performance of stump.

here weight = $\frac{1}{7}$ for all samples & $\boxed{\text{Total Error} = \frac{1}{7}}$

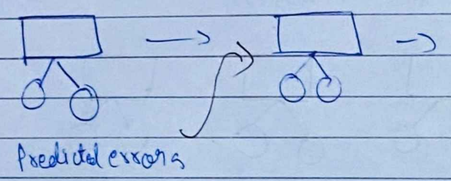
All Thanks To Krish Sir!

② Performance of Stump : $\frac{1}{2} \ln\left[\frac{1 - TE}{TE}\right]$

$= \frac{1}{2} \ln\left[\frac{1 - 1/7}{1/7}\right]$

$= \frac{1}{2} \ln(6) \approx 0.896$

$\boxed{\alpha_1 = 0.896}$ => weight $\left(f = \alpha_1(m_1) + \alpha_2(m_2) \ldots \alpha_n(m_n)\right)$



Predicted errors

③ Update the weights for correctly & Incorrectly classified points

Salary	Credit	Approach	Sample weights	updated wts
<=50k	B	No	1/7 ↓	0.058
"	G	Yes	1/7 ↓	0.058
"	G	Yes	1/7 ↓	0.058
>50k	B	No	1/7 ↓	"
"	G	Yes	1/7 ↓	"
"	N	Yes	1/7 ↑	0.349
<=50k	N	No	1/7 ↓	0.058

For correct classified

weight $\times e^{-\text{Performance of}}$

$\frac{1}{7} \times e^{-(0.896)} = 0.058$

For Incorrect classified

weight $\times e^{\text{Performance of stump}}$

$\frac{1}{7} \times e^{(0.896)} = 0.034$

④ Normalized Weights & Assigning Bins.

Norm weight = $\frac{\text{sum total updated}}{}$

Updated weights	Normalized weights	Bins assignment
0.058	0.08	0 - 0.08
0.058	0.08	0.08 - 0.16
"	0.08	0.16 - 0.24
"	0.08	0.24 - 0.32
"	0.08	0.32 - 0.40
0.349	0.5	0.4 - 0.90
0.058	0.08	0.90 - 0.98
0.697		≈ 1

Note: Normalize the weight so the sum of weights become 1.

Note: Bins assignment make sure to only send those records which are incorrectly classified (error)

- We have created range of bins starting from 0 & adding cumulatively.
- The bins with max. range will be classified as incorrect/error so the model will know this needs to be send to next stump.

⑤ Select data points to send to next stump.

S	credit	Approval	Random	Sample weights.
>50k	N	Yes	0.50	1/6
<=50k	G	Yes	0.10	1/6
>50k	N	Yes	0.60	1/6
>50k	N	Yes	0.75	1/6
<=50k	G	Yes	0.24	1/6
>50k	B	No	0.32	1/6
~~>50k~~	~~N~~	~~Yes~~	~~0.87~~	~~1/6~~

All Thanks to Krish Sir!

⑥ These records will be sent to Next DT stump.

performance stump → 0.65

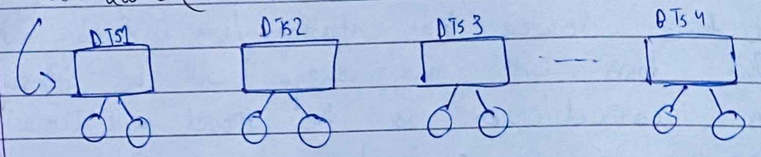
$f_1 = \alpha_1(m_1) + \alpha_2(m_2)$

$\alpha_1 = 0.896$, $\alpha_2 = 0.65$.

Note: From step 2 to step 6 all these will be Repeated.

7 Final Prediction :

Test data (≤50k, G)



Yes	No	Yes	No
$\alpha_1 = 0.896$	$\alpha_2 = 0.650$	$\alpha_3 = 0.24$	$\alpha_n = -0.30$

$$f = \alpha_1(m_1) + \alpha_2(m_2) + \alpha_3(m_3) + \dots \alpha_n(m_n)$$
$$= 0.896\,(\text{Yes}) + 0.65\,(\text{No}) + 0.24\,(\text{Yes}) - 0.30\,(\text{No})$$
$$= 1.136\,(\text{Yes}) + 0.35\,(\text{No}) \Rightarrow O/P = \text{Yes}$$

↳ Performance of say (yes) = 1.136 greater then
↳ Performance of say (No) = 0.350.

In Regression:

Entropy & Gini will not be there instead mean squared error will be calculated.

Instead of Yes/No a continous value will be applied with weights.

All Thanks To Krish Sir!

Page No.
Date 22 11 24

Adaboost classifier — Implementation

Note: For Hyperparameter tuning when puting params in dictionary each sklearn & model name the find parameters related then play with parameters to improvise the predictions.

Practise

GRADIENT BOOSTING ALGORITHM :
(Classification & Regression)

Regression Data set :

Exp.	Degree	Salary	y	R_1	R_2	y	R_3	R_4
2	B.E	50k	75k	-25k	-23	72.7	-22.7	-
3	Master	70k	75k	-5k	-3	74.7	-4.7	-
5	Master	80k	75k	5k	3	-	-	-
6	PHD	100k	75k	25k	20	-	-	-

$\frac{50+70+80+100}{4} = 75k$

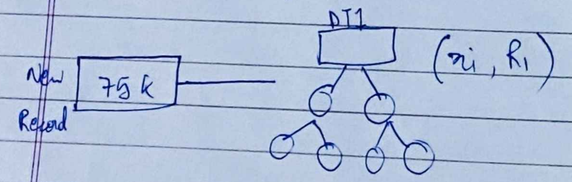
Steps

① Create a Base Model

$\boxed{75}$ Avg. $= \frac{50+70+80+100}{4} = 75$

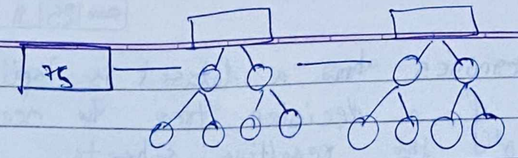
② Compute Residuals, Error (R)

③ Construct a Decision tree.
Consider inputs x_i & O/P R_i



Predicted O/P => $75 + (-23) = 75 - 23 = 52$ (overfitting)

To overcome overfitting we use α (learning rate 0-1)
$\alpha = 0.1$
$75 + \alpha(-23) = 75 + (0.1)(-23)$
$= 72.7$.



$F_n = \alpha_0 h_0(x) + \alpha_1(h_1(x)) + \alpha_2(h_2(x)) + \ldots \alpha_n(h_n(x))$

$$\boxed{f(x) = \sum_{i=0}^{n} \alpha_i h_i(x)}$$ → Final func of Gradient Boosting.

Gradient boosting Algorithm : Implementation classifier:

All the model training will be same we j need to add Gradient boosting.

Hyperparameter Training

~~gb_params = {"max_depth": [5, 8, 15, None, 10]}~~

gradient_Params = {"loss": ['log_loss', 'deviance', 'exponential'],
"criterion": ['friedman_mse", squared_error, 'mse'],
"min_samples_split": [2, 8, 15, 20],
"n_estimators": [100, 200, 300, 1000],
"max_depth": [5, 8, 15, None, 10]

Loss: represents the difference btw. predicted values and true values. The model minimizes this loss during training.
Eg: Log loss, Hinge loss, MSE.

All Thanks To Krish Sir

Criterion: Determines how a dataset is split at each node of a decision tree be measuring the purity of the resulting subsets.
Eg: Gini impurity, Entropy, MSE.

min_samples_split: Defines the min. no. of samples required to split a node.

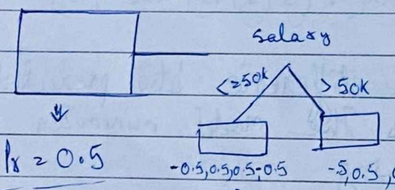
n_estimators: Represents the no. of individual base models in ensemble.

Xgboost ML Algorithm (classification)

Dataset $\lambda = 0.5$

Salary	Credit	Approval	R1	$\hat{y}$	R2
<=50k	B	0	-0.5	0.52	-0.48
<=50k	G	1	0.5	0.508	0.492
<=50k	G	1	0.5	–	–
>50k	B	0	-0.5	–	–
>50k	G	1	0.5	–	–
>50k	N	1	0.5	–	–
<=50k	N	0	-0.5	–	–

(1)



$P_R = 0.5$

similarity weight (LC) $= \frac{\Sigma (\text{Residual})^2}{\Sigma P_R(1-P_R)}$

$= \frac{[(-0.5+0.5+0.5-0.5)^2]}{[0.5(1-0.5)+0.5(1-0.5)+0.5(1-0.5)+0.5(1-0.5)]}$

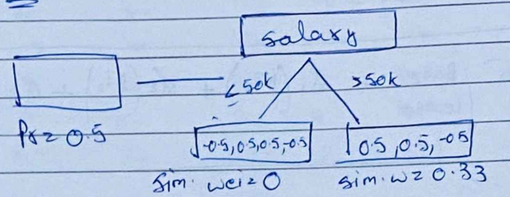
$= 0$

similarity weight (RC) $= \frac{(-0.5+0.5+0.5)^2}{0.75}$

$= \frac{0.25}{0.75} = \frac{1}{3} = 0.33$

SW (root) $= \frac{0.25}{1.75} = 0.14$.

Gain $= 0 + 0.33 - 0.14 = 0.21$



$P_R = 0.5$

sim. wei = 0

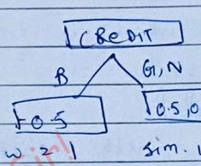
sim. w = 0.33

sim w (LC) $= \frac{0.25}{0.25} = 1$

sim w (RC) $= \frac{0.25}{0.75} = 0.3$

Gain $= 1 + 0.33 - 0$

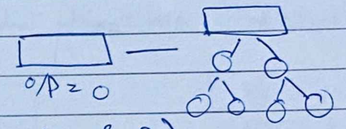
$= 1.33$



sim. w = 1

sim. w = 0.33

Final O/P



O/P = 0

for test data: (<=50k & B)

* We use log of odds => $\log(\text{odds}) = \log\left(\frac{P}{1-P}\right)$

for base model $\log\left(\frac{0.5}{0.5}\right) = \log(1) = 0$.

$= \sigma(0 + \alpha(1))$ → similarity weight

sigmoid Activation $\alpha = 0.1$

$= \sigma((0 + (0.1)(1))$

$= \frac{1}{1+e^{-0.1}} = 0.52$

All Thanks To Krish Sir

Second record o/p $= \sigma(0 + \alpha(0.33))$

$= \sigma(0 + 0.1(0.33))$

$= \frac{1}{1 + e^{-0.033}} = 0.508$

$$\boxed{\text{Final function} = \sigma\left(\text{Base learner} + \alpha_1(DT_1) + \alpha_2(DT_2) + \alpha_3(DT_3)\right)}$$

Steps:

1. Construct a base Model
2. Construct a Decision Tree with root
3. Calculate similarity weight $= \frac{\sum(\text{Residuals})^2}{\sum p_i(1-p_i) + \lambda}$

   cover value — hyperparameter
4. Calculate gain

Extreme Gradient Boosting

Xgboost Regresson ML Algorithm:

Dataset {Regression}

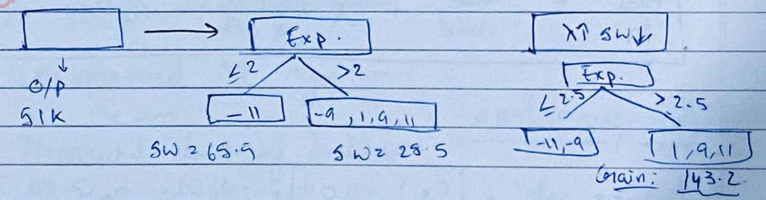
Exp	Gap	salary	R_1	$\hat{y}$	R_2
2	Yes	40k	-11	50	10
2.5	Yes	42k	-9	50	12
3	No	52k	1	51.5	0.5
4	No	60k	9	51.5	8.5
4.5	Yes	62k	11	52.1	9.9

$\approx 51k$

similarity weight $= \frac{\sum(\text{Residual})^2}{\sum p_i(1-p_i) + \lambda}$ → for Regression $= \frac{\sum(\text{Residual})^2}{\text{No. of residual} + \lambda}$

Steps

1. Create a Base Model 2. Residual Computation 3. Construct DT1 using $\{x, R_i\}$



4. Calculate SW ⇒ $\lambda = 1$ $\quad SW(LC) = \frac{121}{1+1}$

$SW(RC) = \frac{(-9 + 1 + 9 + 11)^2}{4+1} = \frac{144}{5} = 28.5 \quad = \frac{121}{2} = 65.5$

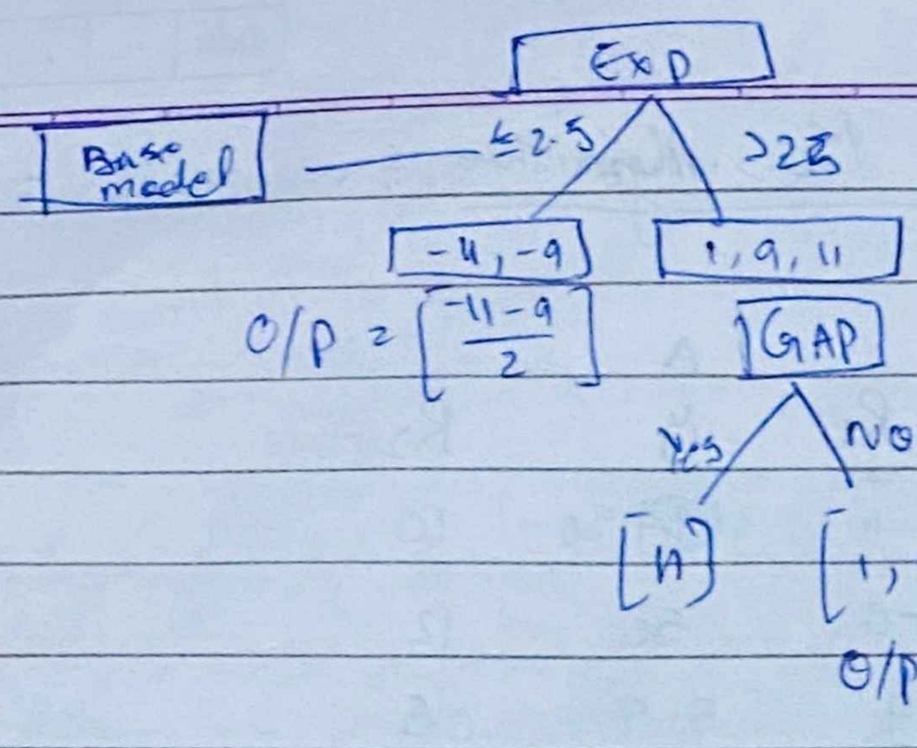
5. Calculate Gain:

$SW(\text{Root node}) = \frac{1}{6} = 0.16$

$\text{Gain} = 65.5 + 28.5 - 0.16 = 93.24$

The threshold with more gain will be selected

All Thanks To Krish Sir



α = learning rate

$\alpha = 0.1$

In regression we take avg. not similarity weight (log odds)

Final output :

$[51 + (0.1)(-10)] = 51 - 1 = 50$

$51 + (0.1)(5) = 51.5$

$51 + (0.1)(11) = 52.1$

$$\boxed{\text{Final fn.} = \text{base model} + \alpha_1(\text{average of O/P}) + \alpha_2(\text{avg}) + \alpha_3(\text{avg})}$$

All Thanks To Krish Sir!

Hyperparameters

'learning_rate' : [0.1, 0.01],
'max_depth' : [5, 8, 12, 20, 30],
'n_estimators' : [100, 200, 300],
'colsample_bytree' : [0.5, 0.8, 1, 0.3, 0.4]

Cols sample_bytree : How many features to take in every tree.
(0.8 means 80%)

Page No.
Date 26 11 24

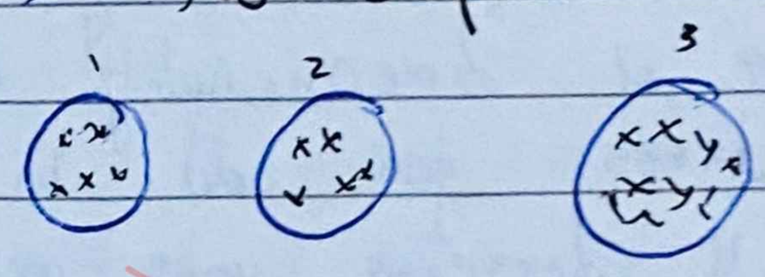
UNSUPERVISED MACHINE LEARNING :

The difference btw. supervised & unsupervised is that in supervised machine learning we predict something or get output but in unsupervised we do clustering.

Unsupervised ML → Clustering (group your data into similar clusters)

Age Experience Salary → No output.

1 { - - -
- - - }3
2 { - - -



Real life example : Product : Data Salary spending - score
↳ new product

Unsupervised ML Algorithm :

1. K means
2. Hierarchal clustering
3. DB Scan clustering
4. Silhoutte Scoring

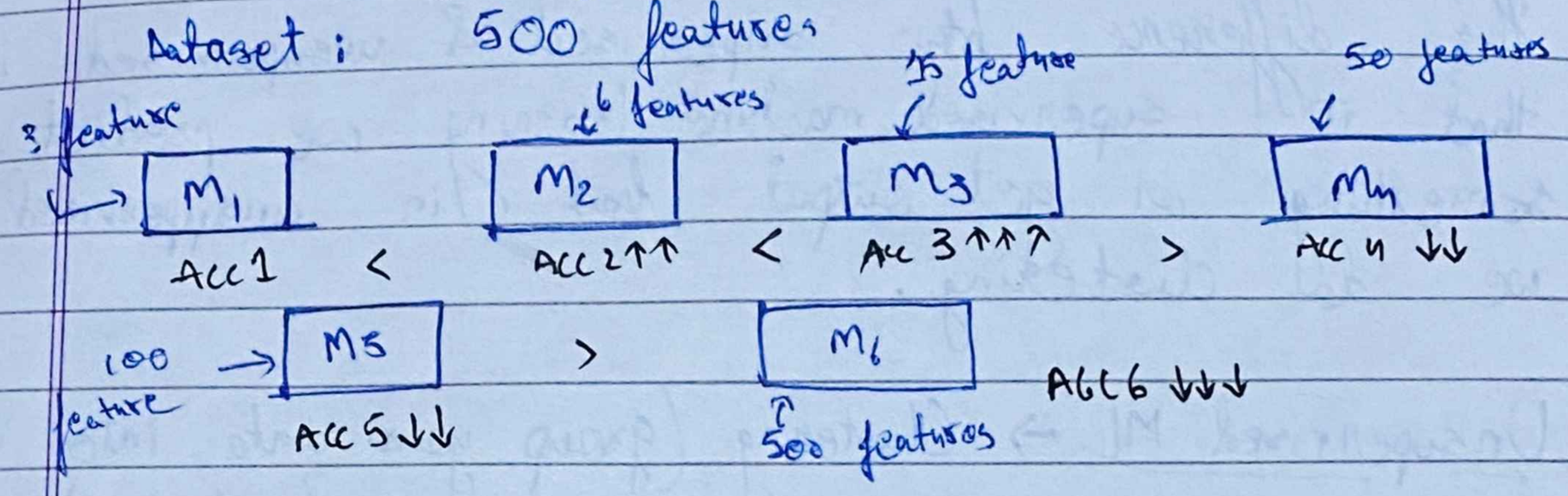
} validate

All Thanks To Krish Sir!

Principal Component Analysis (PCA):

Curse of Dimensionality:

Dataset : 500 features (dimensions)



Model performance degrade:

Curse of dimensionality states that the more no. of features you add in a model, beyond needed, it will decrease your accuracy & make your model overfit & create a lot of confusion.

Two diff. ways to remove curse of dimensionality.

1. Feature selection → Imp. features
2. Dimensionality Reduction (PCA) → feature extraction.

All Thanks To Krish Sir!

Page No.
Date

Feature Selection Vs Feature Extraction.
↳ Dimensionality Reduction

① Why Dimensionality Reduction?

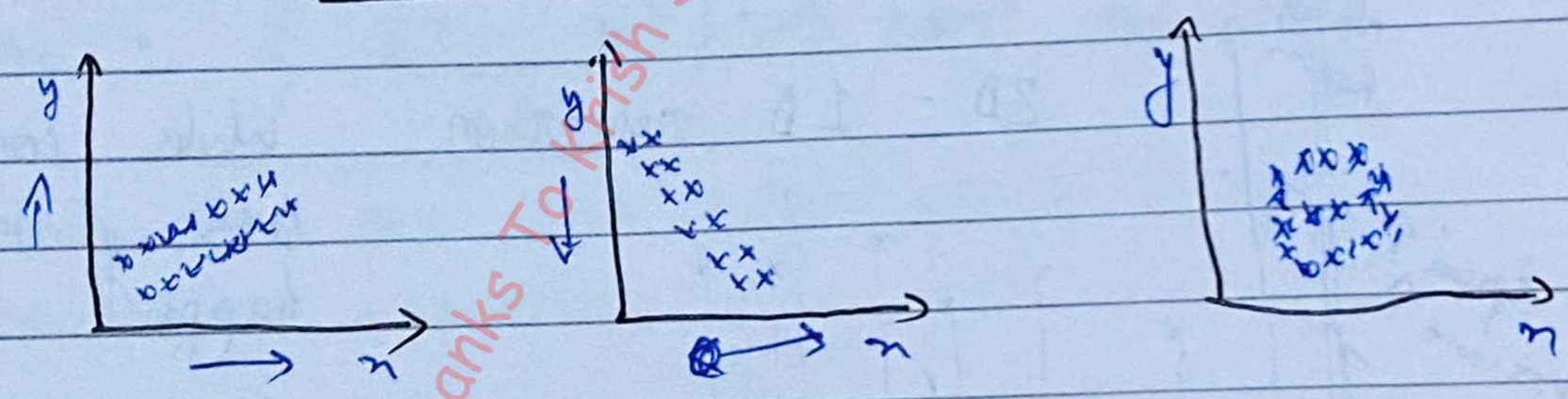
* Prevent → curse of Dimensionality
* Improve the performance of the model
* Visualize the data → understand the data.

Feature Selection

I/P	O/P	
x	y	+ve
-	-	
-	-	
-	-	
-	-	

⇒ $\boxed{x\uparrow \quad y\uparrow \qquad x\downarrow \quad y\downarrow}$ $\boxed{x\downarrow \quad y\uparrow \qquad x\uparrow \quad y\downarrow}$ (−ve)

No relationship btw x & y



$$Cov(x,y) = \sum_{i=1}^{n} \frac{(x_i - \bar{x}) * (y_i - \bar{y})}{N-1} = +ve, -ve \approx 0 \text{ (No relationship)}$$

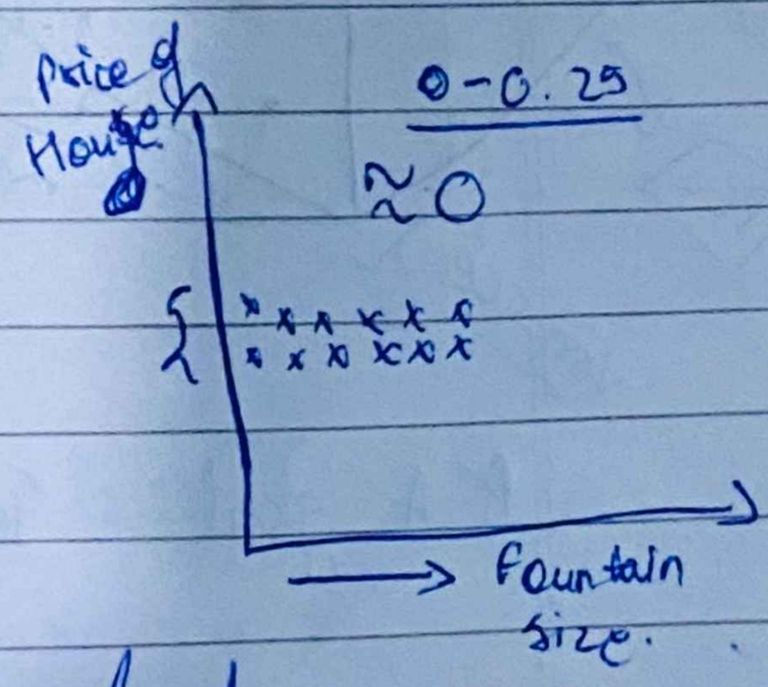
$$\text{Pearson Correlation} = \frac{Cov(x,y)}{\sigma_x \times \sigma_y} = \underline{-1 \text{ to } 1}$$

The more toward the value of +1 the more +ve correlated X & Y is & vice-versa.

Dataset Housing:

House size	Fountain size	Price
-	-	-
-	-	-

↳ Drop.



Here we drop less correlated feature.

All Thanks To Krish Sir!

Feature Extraction :

2 features → 1 feature
Dimensionality Reduction.

Room size No. of Rooms Price

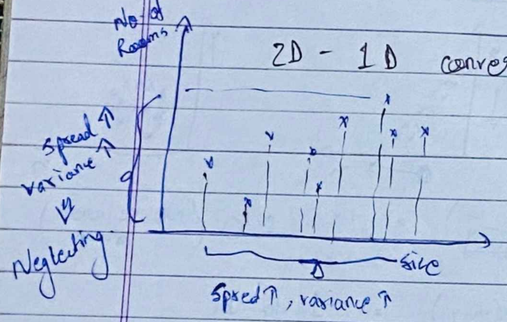
⇓ ⇓ Transformation to extract New features

House size Price.

PGA Geometric Intuition :

Housing Dataset :

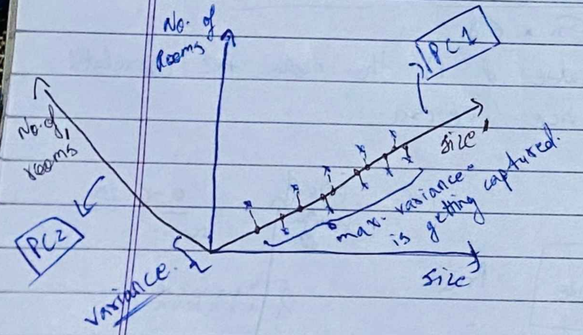
Size House	No. of Room	Price



2D - 1D conversion. While converting a major loss of information (No. of Rooms) happen

Feature Extraction

2 Dimension
PC1, PC2
Eigen decompositions matrix } var(PC1) > var(PC2)



2D - 1D
Much information is not lost.

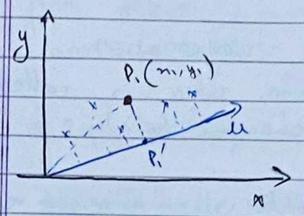
PCA goal : To get the best principal component which captures max. variance.

All Thanks To Krish Sir

Page No.
Date 27 11 24

Maths Intuition behind PCA algorithm :

1. Projection
2. Cost function → Variance



$$\boxed{\text{Proj}_{P_i} u = \frac{P_i \cdot u}{\|u\|}}$$

$\|u\| = 1$ → unit vector.

P_i' ← $\boxed{\text{Proj}_{P_i} u = P_i \cdot u}$ ⇒ Scalar value

$x_0', x_1', x_2', x_3', x_4' \ldots\ldots x_n'$.

$$\boxed{\text{Max. Variance} = \sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{n}}$$ ⇒ cost function

Goal : Find the best unit vector which captures maximum variance.

Eigen vectors And Eigen values

1. Covariance Matrix btw. features
2. Eigen vectors and eigen values will found out from this covariance matrix
3. Eigen vector → Eigen value → magnitude of the Eigen vector → capture the max. variance

Eigen vectors And Eigen Values (Linear Transformation)

$$\boxed{A \times V = \lambda \times V}$$

$\Downarrow$

eigen value

eigen vector is just a linear transformation
The vector jisko direction same rahta is called eigen vector, lekin magnitude change hota rha

Eigen value is used to calculate value of eigen vector and it measures variance explained

PCA steps :

- Get original - data
- Calculate Covariance Matrix
- Calculate Eigen Vectors
- Sort Eigen Vectors by Eigenvalue
- choose N largest eigenvalue / variance
- Project original data to eigen vectors

$$cov(x,y) = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{N-1}$$

$\boxed{x, y}$ z $\quad cov(x, x) = Var(x)$

↳ x'

Covariance Matrix =

	x	y
x	Var(x)	Cov(x,y)
y	Cov(x,y)	var(y)

PCA - Implementation :

Import all important libraries

```
from sklearn.datasets import load_breast_cancer
cancer_dataset = load_breast_cancer()

df = pd.DataFrame(cancer_dataset['data'], columns = cancer_dataset['feature'])

## standardization
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(df)
scaled_data = scaler.transform(df)

## Applying

from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
data_pca = pca.fit_transform(scaled_data)

# Explain variance
pca.explained_variance
```

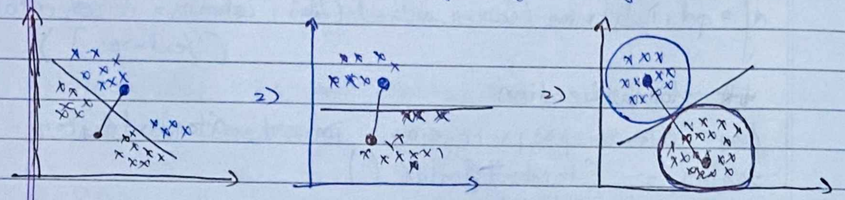
PC1 > PC2

```
# Plotting
plt.figure(figsize = 8,6))
plt.scatter(data_pca[:,0], data_pca[:,1],
c = cancer_dataset['target'], cmap = 'plasma')

plt.xlabel('First principal component')
plt.ylabel('Second principal component')
```

All Thanks To Krish Sir

K- Means Clustering Algorithm:

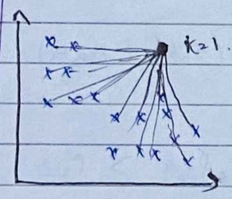
1. Initialize some K]-centroid
2. Points that are nearest to the centroid group them
3. Move the Centroids (By taking average)



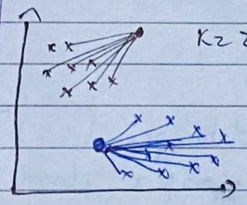
How do we select the K value?

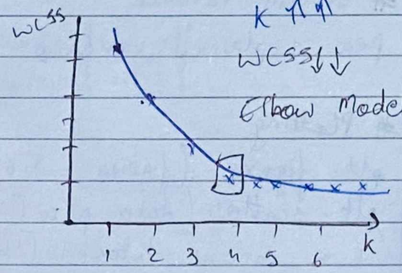
WCSS = within cluster sum of squares

Initialize K=1 to 20



$$WCSS = \sum_{i=1}^{n} \left(\text{distance between points to nearest centroid} \right)^2$$



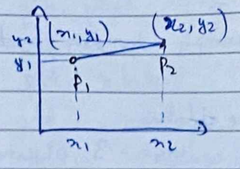


K ↑↑
WCSS ↓↓
Elbow Model

K=3

As we increase k value the value of WCSS decreases and at some point it does not make any difference
↳ After elbow point.

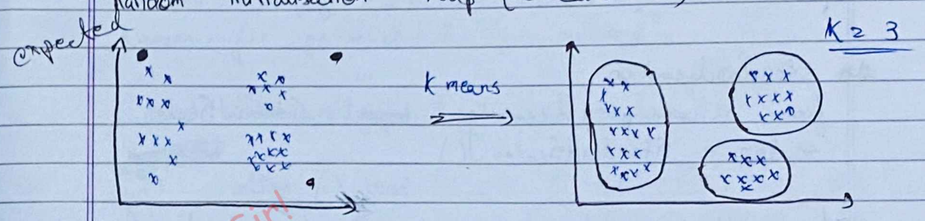
Euclidean Distance:

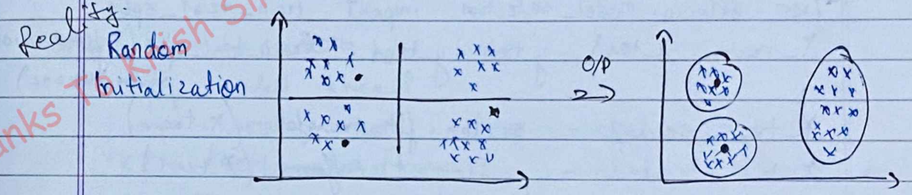


Euclidean dist. = $\sqrt{(x_2-x_1)^2 + (y_2-y_1)^2}$

Manhattan dist. = $|x_2-x_1| + |y_2-y_1|$

Random Initialisation Trap (Kmeans++)





To avoid this misclustering we use Kmeans++

Kmeans ++ initialization:

It is used in such a way that it is completely at a farther distance btw. each other.

All Thanks To Rajesh Sir!

K-means Clustering - Implementation :

Import all imp. libraries (numpy, pandas, matplotlib)

```
from sklearn.datasets import make_blobs
x, y = make_blobs (n_samples=1000, centers = 3, n_features = 2)

plt.scatter (x [:,0], x [:,1], c=y)
```

standardization

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

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split (X, y, test_size=0.33, random_state = 42)

X_train_scaled = scaler.fit_transform (X_train)
X_test_scaled = scaler.transform (X_test)

from sklearn.cluster import KMeans
```

Elbow method to select K value.

```
wcss = []
for k in range (1,11):
    Kmeans = KMeans (n_clusters = k, init = "k-means++")
    Kmeans.fit (X_train_scaled)
    wcss.append (Kmeans.inertia_)
```

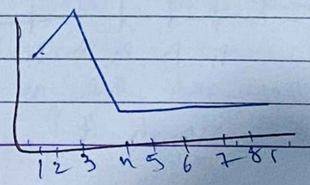
```
# plot elbow curve
plt.plot (range (1,11), wcss)
plt.xticks (range (1,11))
plt.xlabel ("number of Clusters")
plt.ylabel ("wcss")
plt.show ()

Kmeans = KMeans (n_clusters = 3, init = "k-means++")
Kmeans.fit_predict (X_train_scaled)

y_pred = Kmeans.predict (X_test_scaled)

plt.scatter (X_test [:,0], X_test [:,1], c = y_pred)
```

Knee locator

```
pip install kneed
from kneed import KneeLocator
Kl = KneeLocator (range (1,11), wcss, curve = "convex", direction = "decreasing")
Kl.elbow
```

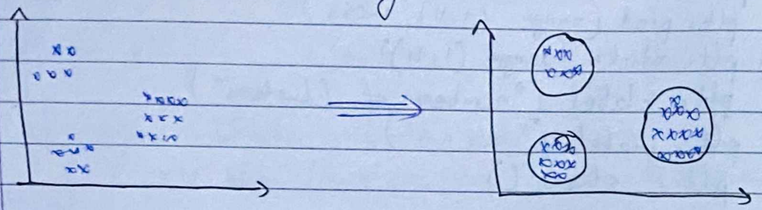
silhouette score

```
from sklearn.metrics import silhouette_score
silhouette_coefficients = []
for k in range (2,11):
    Kmeans = KMeans (n_clusters = k, init = "k-means++")
    Kmeans.fit (X_train_scaled)
    score = silhouette_score (X_train_scaled, kmeans.labels_)
    silhouette_coefficients.append (score)
```



All Thanks To Krish Sir.

Hierarichal Clustering

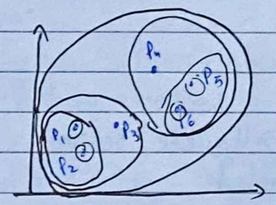


2 types

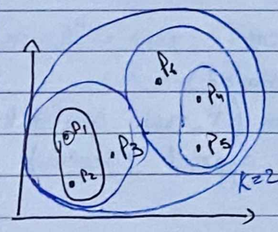
1) Agglomerature → Addition / combining } Geometric
2) Divisive

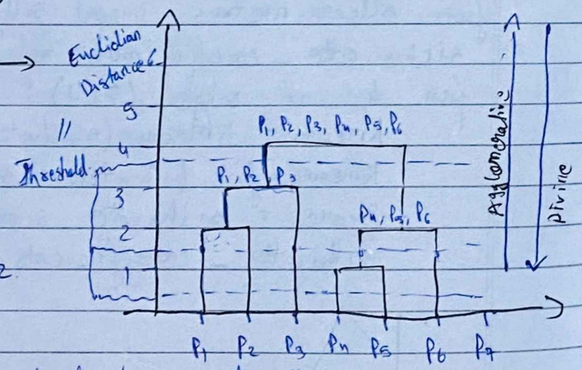
Steps :

1. For each point initially will consider it as a seperate cluster
2. Find the nearest point and create a new cluster
3. Keep on doing the same process until we get a single cluster



Dendrogram:





Select the longest vertical line such that no horizontal line passe through it

Hierarichal clustering - Implementation

All Thanks To Krishna Sir

Import IRIS dataset

```
iris = datasets.load_iris()
iris_data = pd.DataFrame(iris.data)
iris_data.columns = iris.feature_names

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(iris_data)
```

Apply the PCA

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit_transform(X_scaled)
plt.scatter(pca_scaled[:,0], pca_scaled[:,1],
            c=iris.target)
```

Agglomerative Clustering

To construct a dendrogram

```
import scipy.cluster.hierachy as sc
```

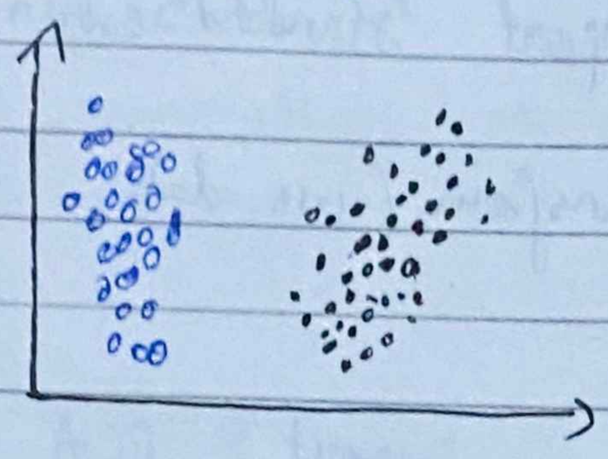
plot the dendrogram

```
plt.figure(figsize=(20,7))
plt.title("Dendrograms")
```

Create dendrogram

```
sc.dendrogram(sc.linkage(pca_scaled, method='ward'))
plt.title('Dendogram')
plt.xlabel('Sample Index')
plt.ylabel('Euclidean Distance')
```

```
from sklearn.cluster import AgglomerativeClustering
cluster = AgglomerativeClustering(n_clusters=2, affinity='euclidean',
                                  linkage='ward')
cluster.fit(pca_scaled)
cluster.label_
plt.scatter(pca_scaled[:,0], pca_scaled[:,1], c=cluster.label_)
```



K means vs Hierarichal clustering.

Scalability And Flexibility.

1. Dataset size → Huge → K means
   small → Hierarichal clustering.

2. K mean → Numerical data
   Hierarichal clustering → variety of data

3. Centroids → Elbow method → No. of centroids difficult to find
   H.C → No. of clusters easy.

All Thanks To Krish Sir!

Density Based.

DB Scan Clustering :

- • → Core point min. pts = 4 , ϵ = radius
- • → border point
- ⊙ → Outlier

Core point :

① No. of points within the ϵ should be greater $\geq$ min. points

Border point :

② no. of data points within this radius will be less than min. points.

Outlier :

When no point exists except the ϵ point.

→ Even with outlier it will handle wonderfully.

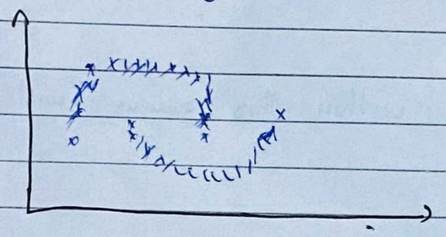
→ can also handle non-linear clustering.

Advantages	Disadvantages
1) No need to specify clusters	1) Non deterministic for Border points
2) Finds clusters of any shape	2) Difficult with varying Densities
3) Robust to Noise	3) parameter sentivity
4)	

All Thanks To Kush Sir!

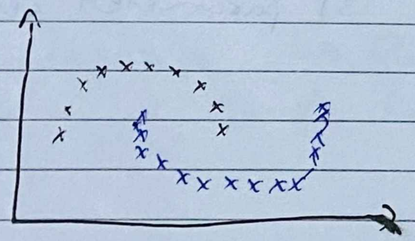
DBScan Implementation

```
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_moons
import matplotlib.pyplot as pl

X, y = make_moons (n_sample = 250, noise = 0.05)
plt.scatter ( X [:,0] , X [:,1] )
```



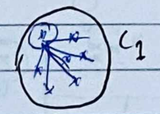
Feature Scaling

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

X_scaled = scaler.fit_transform (X)
dbscan = DBSCAN (eps = 0.3)
dbscan.fit (X_scaled)

plt.scatter (X [:,0] , X [:,1] , c = dbscan.label_ )
```



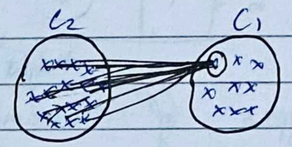
Silhouette Clustering:

1) $a(i) = \dfrac{\Sigma \text{ distance}}{\text{Total no. of points}}$



a(i)

2) $b(i) = \dfrac{\Sigma \text{ distance to } C_2}{\text{Total no. of points}}$



$a(i) << b(i)$

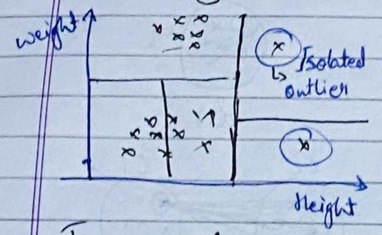
3) Silhouette scoring. (It ranges btw. -1 to 1.)

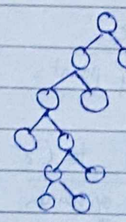
$$s(i) = \frac{b(i) - a(i)}{\max. \{a(i), b(i)\}}$$

All Thanks To Krish Sir!

Anomaly Detection [To detect Outliers]

↳ play a imp. role.





↳ Isolated as a leaf node.

Types of anomaly detection:

① Isolation Forest : [Decision Trees]
↳ only unsupervised whole DT used.

Anomaly score :

$$\boxed{S(x,m) = 2^{-\frac{E(h(x))}{c(m)}}}$$

m = no. of data points.
x = Data point.

$E(h(x))$ = Avg. search depth for x from the isolated tree.
$c(m)$ = Avg. depth of $h(x)$

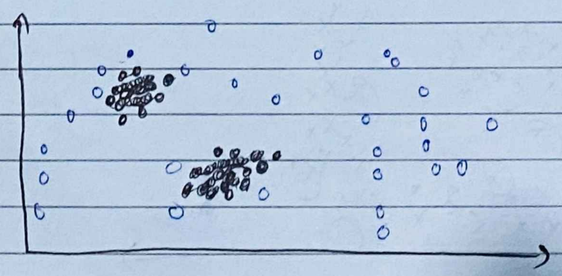
$E(h(x)) << c(m) \Rightarrow S(x,m) \approx 1$
↳ Anomaly score = outliers

[threshold ≥ 0.5] $E(h(x)) >> c(m) \Rightarrow S(x,m) \approx 0.5$
↳ normal point.

All Thanks To Krishna Sir

Implementation

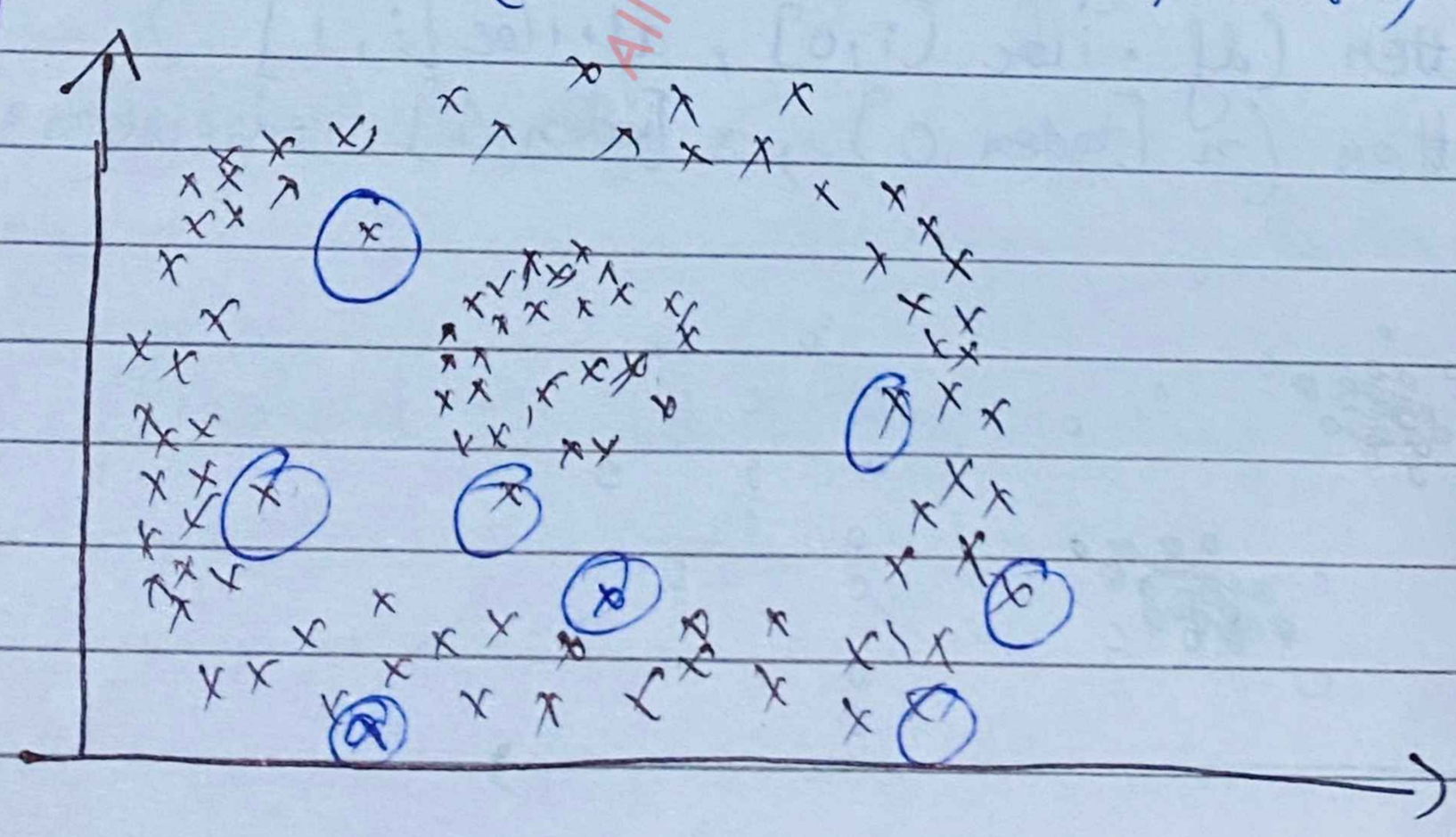
```
import pandas as pd
df = pd.read_csv('healthcare.csv')
df.head()

import matplotlib.pyplot as plt
plt.scatter(df.iloc[:,0], df.iloc[:,1])

from sklearn.ensemble import IsolationForest
clf = IsolationForest(contamination=0.1)
clf.fit(df)
predictions = clf.predict(df)

import numpy as np
~~abn_ind = np.where(prediction<0)~~
x = df.value
index = np.where(prediction < 0)
plt.scatter(df.iloc[:,0], df.iloc[:,1])
plt.scatter(x[index,0], x[index,1], edgecolors='r')
```



Page No.
Date

DBSCAN Clustering Anomaly Detection

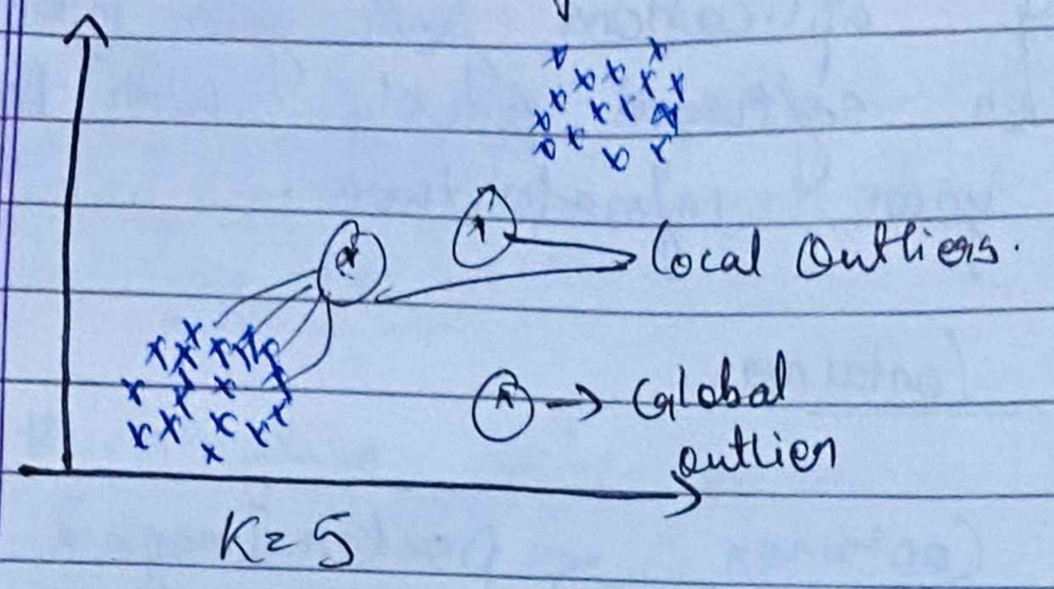
```
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_moons
import matplotlib.pyplot as plt
from sklearn.datasets import make_circles

X, y = make_circles(n_samples=750, factor=0.3, noise=0.1)

plt.scatter(X[:,0], X[:,1], c=y)
from sklearn.cluster import DBSCAN

dbscan = DBSCAN(eps=0.10)
dbscan.fit_predict(X)
dbscan.labels_

plt.scatter(X[:,0], X[:,1], c=dbscan.labels_)
plt.scatter(X[:,0], X[:,1], c=y)
```



All Thanks To Krish Sir!

Page No.
Date

Local Outlier factor Anamoly Detection:



① Local Outlier
② Global Outlier } [LOF] → log score

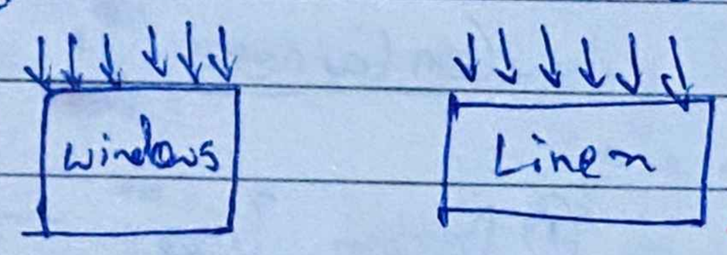
K Nearest Neighbour
⇓
Local Density.

Docker Series:

① What is Docker? ② What are Containers? ③ Container Vs Virtual machine
④ Docker image Vs contatiner ⑤ Prac. Implementation of docker

Why Containers?

Anaconda Libraries mismatch



→ Dev Enviroment ⇒ Q/A.
On other server modules may not work.

Developer A ⇄ Developer B

Containers.

* A way to package application with all the necessary dependiencies And configuration

* Portable artifact, easily share and move this package to any environment.

* Makes development & deployment more easy and efficient.

Now we put app in container the Q/A runs app in container

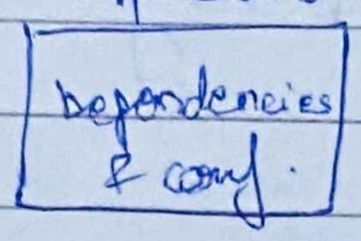
All Thanks To Krish Sir!

Page No.
Date

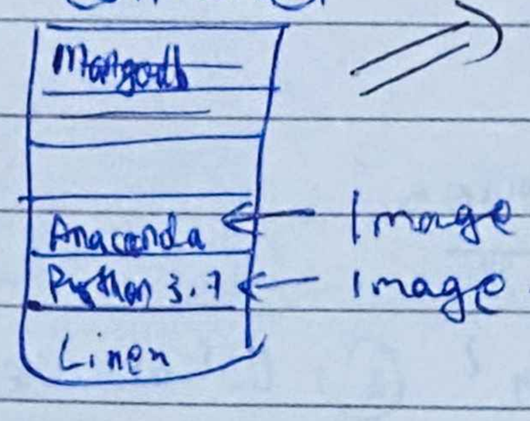
Docker: Open platform for developing, shipping and running aplications from your infrastructure so you can deliver software quickly. With Docker you can manage your infrastructure.

Docker Image And Containers

Application



Container

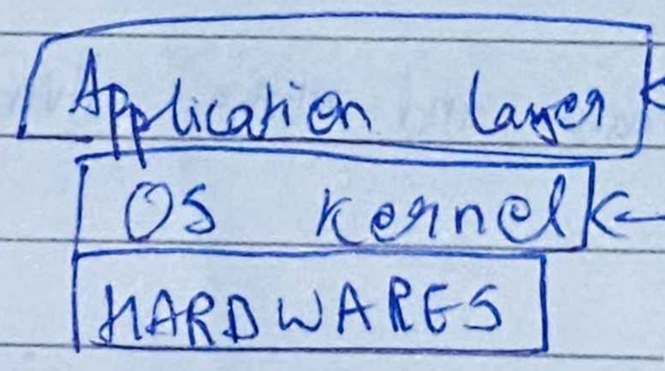


⇒ Docker Image:

Docker Image $\xrightarrow{\text{Run}}$ [• Application environment] ⇒ container.

Docker Image {small}	Container
① Package or Artifact	① Docker Image → Run
② More or share this Artifact	② start the Application
	③ Create a container ⇒ Environment ⇓ Running

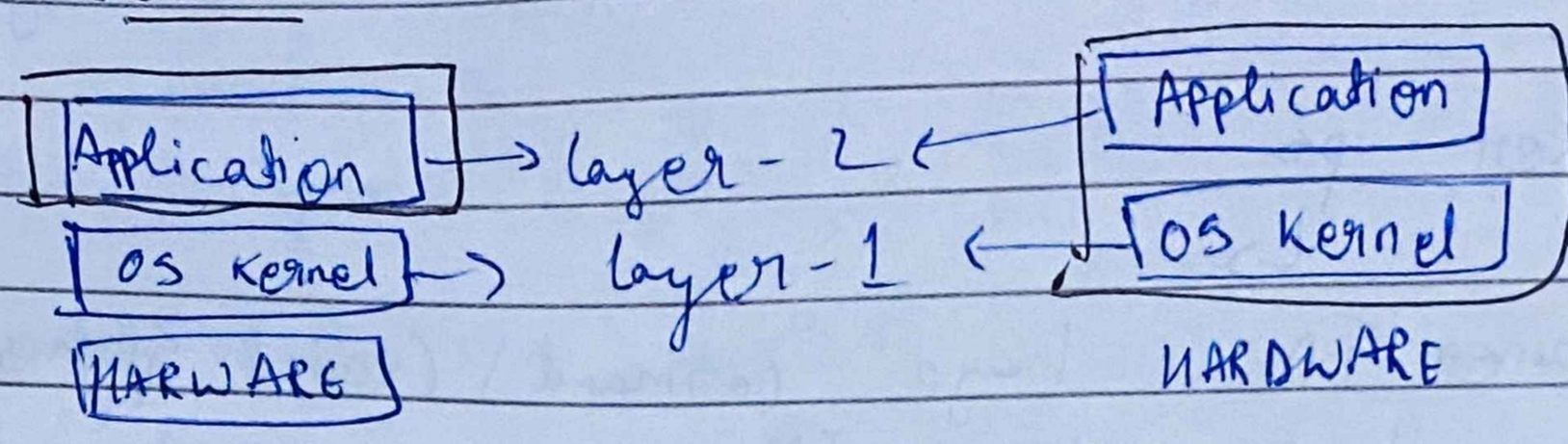
Docker Vs Virtual Machine.



All Thanks To Krish Sir!

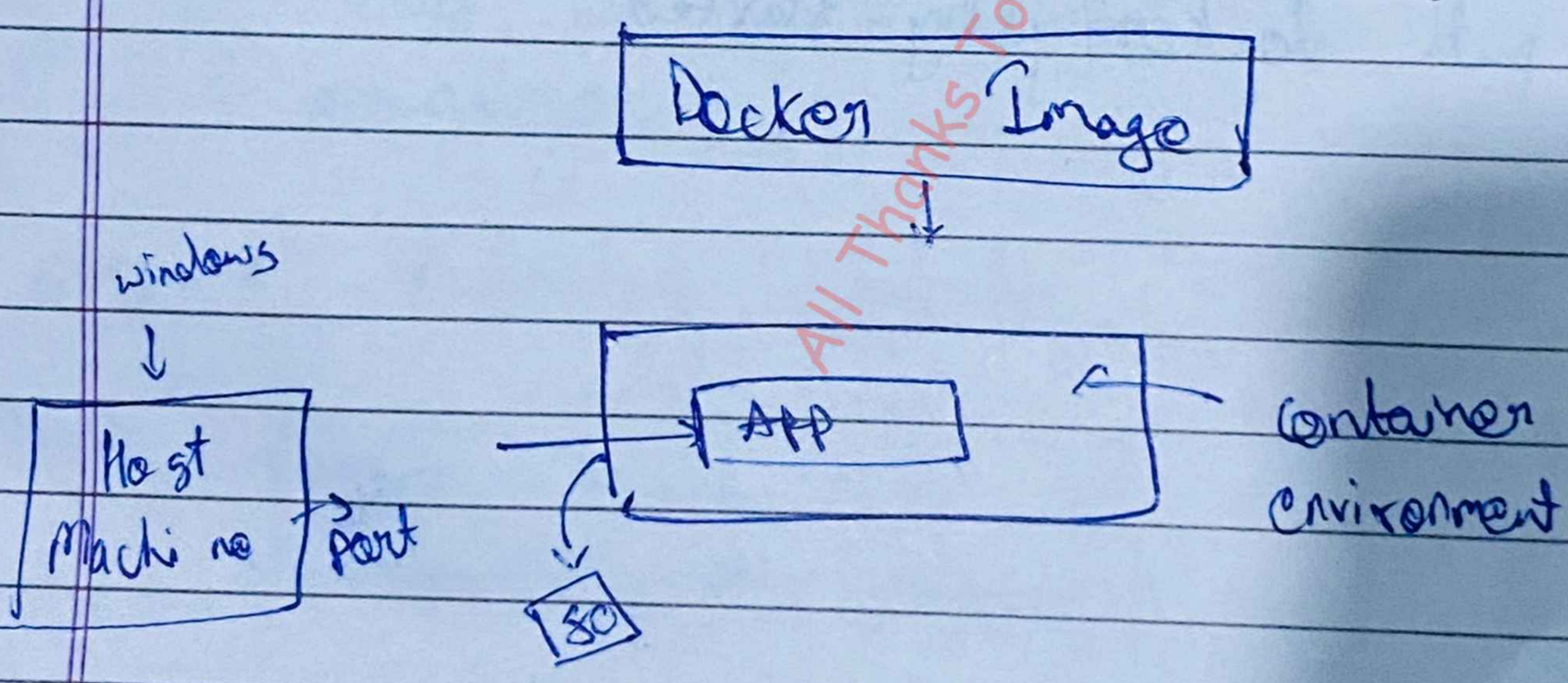
Page No.
Date

Dockers Vs Virtual Machine.



# virtualise Application & communicate to OS kernel.	Both Application with OS Kernel can be communicated.
• Docker image size is usually smaller (MB)	size huge (GB)
• Docker containers start & run much faster	VM is comparatively slower.
• Docker image → [Compatibility Issue]	# Imp. Install VM on any OS

Container Port And Host Port



127.0.0.1:80.

All Thanks To Krish Sir!