Linear and Logistic Regression HandWritten Notes

- Darshan R M

"Learn, Share, and Collaborate"

15

Week -15

linear and logistic regression

* 1

Supervised ML → Regression ⇒ o/p → Continuous
→ Classification ⇒ o/p → discrete

* Simple linear regression

Dataset:

i/p (independent) weight	o/p (dependent) (Continuous) height	BMI
74	170	
80	180	
75	175.5	
..	..	
=	=	

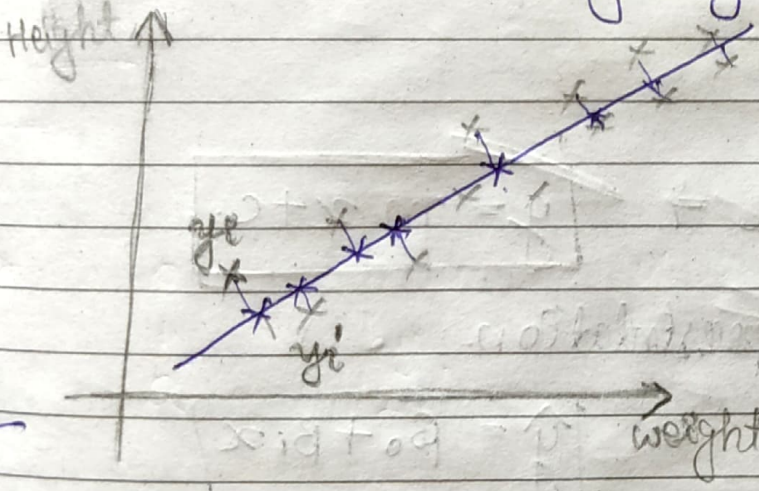
Train

New i/p weight ⇒ [Model] ⇒ o/p height

* Simple linear regression
  ↳ we have 1 input and 1 output

* Multiple linear regression
  ↳ we have multiple input and 1 output

* Geometrical intution
  ↳ gives main aim what we are going to do and how.



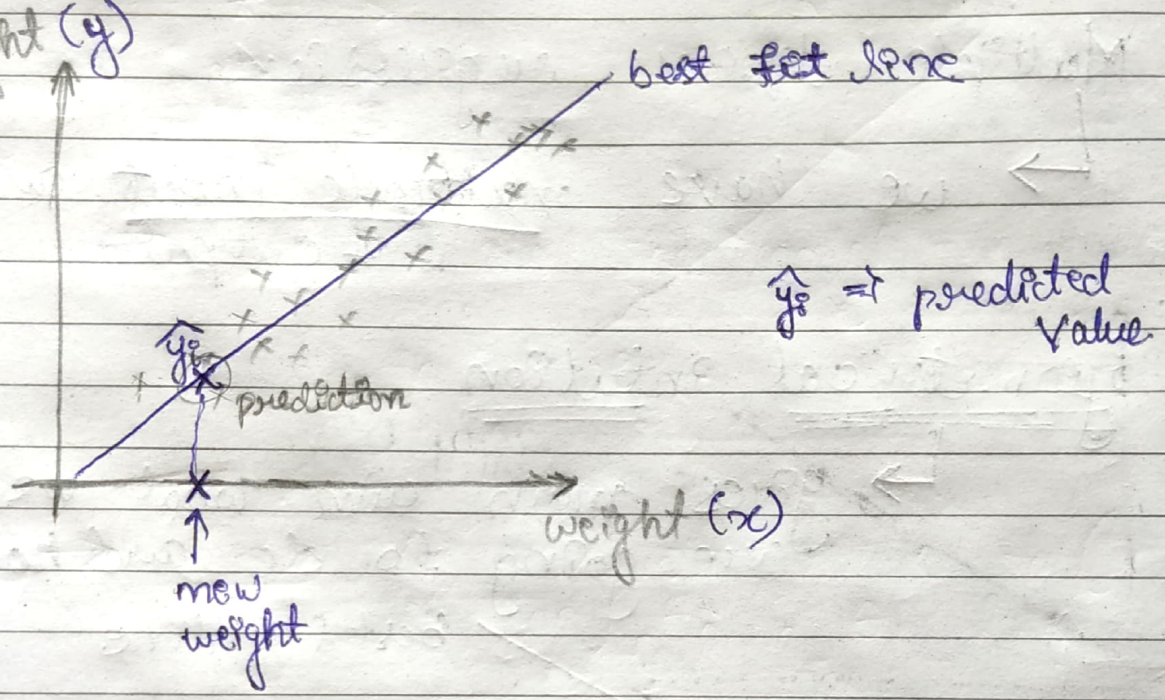
* y_i ⇒ actual point
* y_i' ⇒ predicted point
  y_i

↳ * we try to find out best fit line.
  ↓
such that the summation of the errors is least.
  ↓↑
between actual and predicted point

Σ Errors → minimal

* why to find best fit line.

↳ Because when ever the new weight comes we have to predicte height.



best fit line ⇒ $y = mx + c$.

other representation.

$$\hat{y} = \beta_0 + \beta_1 x$$

researcher → $h_\theta(x) = \theta_0 + \theta_1 x$

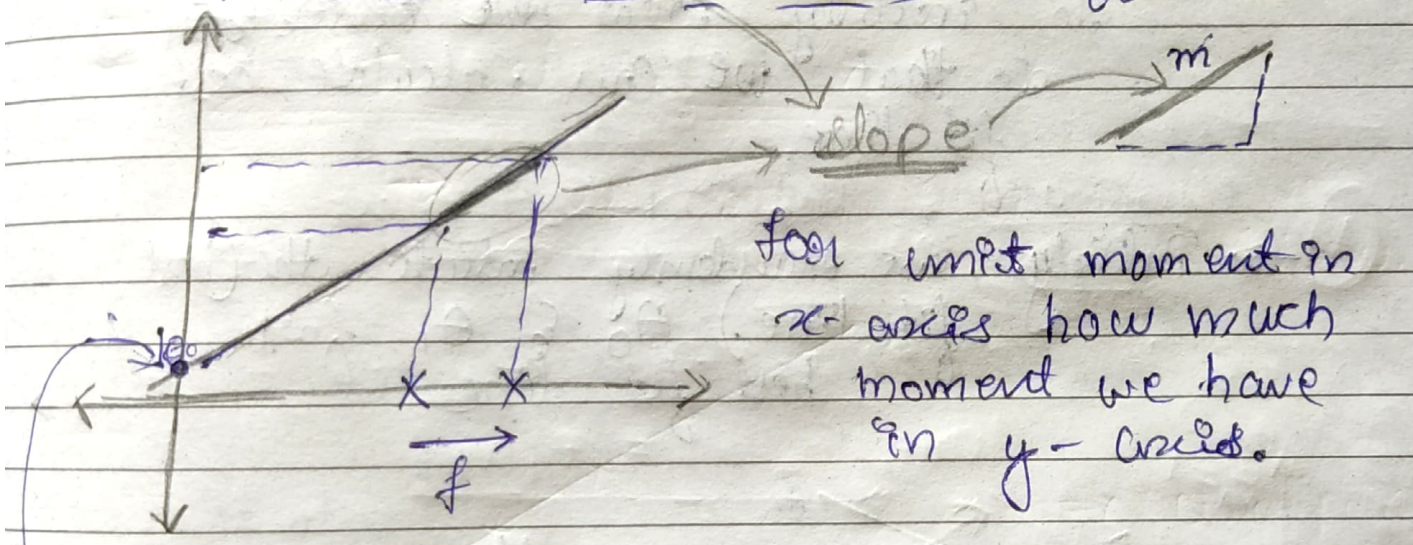
↓

because ml is also treated as hypothesis testing

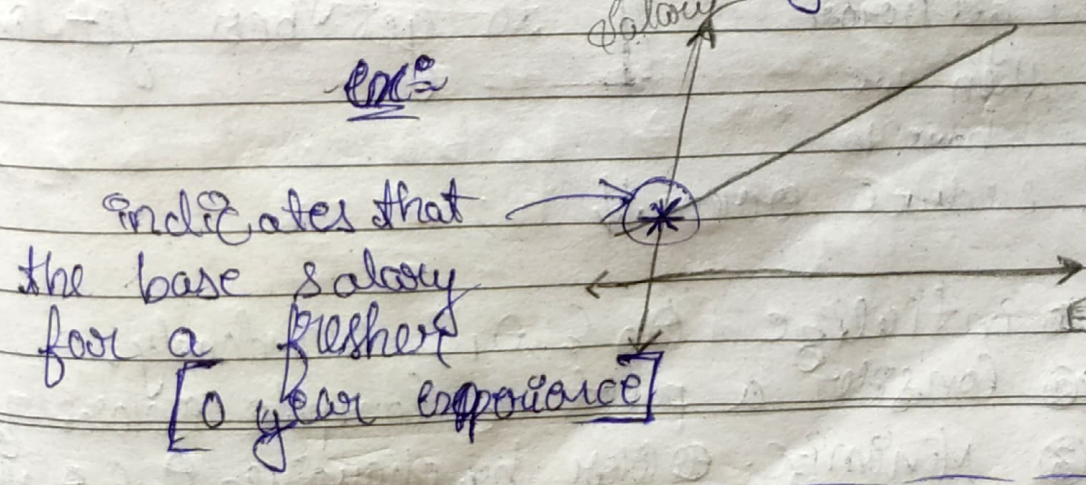
Error $\{ y_i - \hat{y}_i \}$ actual - predicted

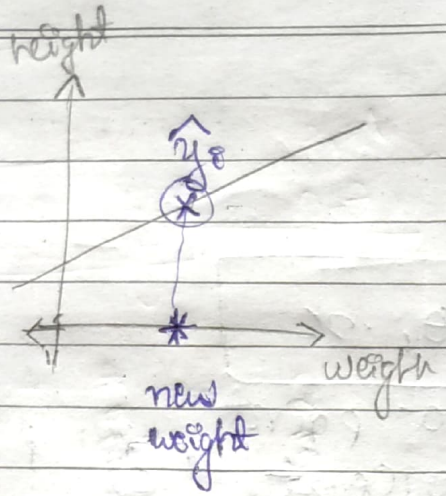
* $h_\theta(x) = \theta_0 + \theta_1 \cdot x$

where, x_i = data point
θ_0 = Intercept. (c)
θ_1 = slope (or) Coefficient. (m)



θ_0 ⇒ intercept ⇒ when $x=0$ where do our line intercept at y-axis.
[c]

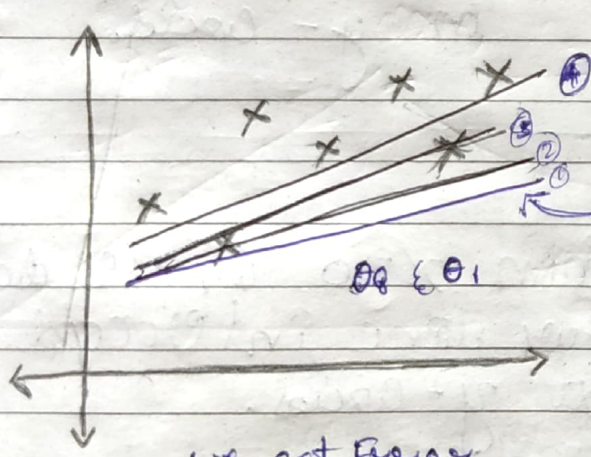




For trained model for any new weight it is calculated like this

* But, to train the model we have to initially treat the θ_0 & θ_1, so that we can calculate best fit line.

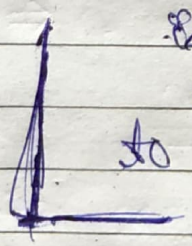
→ Intialy, we randomly initiate the (best fit line) θ_0 & θ_1 for best fit line



initally.

we get Error then we have to reduce error

So now our aim is to continously change θ_0 and θ_1 to get best fit line.



Steps → ① Intialize θ_0 & θ_1 randomly.
② Consider a point & Compute error.
③ Minimize error by changing θ_0 & θ_1.
④ eg

2 important things about Optimization

① Cost Function [Mean squared error]:

↳ To find the summation of error we use cost function.

reduce

$$J(\theta_0, \theta_1) = \frac{1}{n} \sum_{i=1}^{n} [y_i - h_\theta(x)]^2$$

(y_i: actual, $h_\theta(x)$: predicted)

because we change values of θ_0 and θ_1

⇓

mean square error

* we need to "minimize" $J(\theta_0, \theta_1)$ [cost function] to get the best fit line

where, n = number of data points

y_i = actual value

$h_\theta(x)$ = predicted value.

Final aim: To get best fit line.

by reducing value of cost function

→ This entire process is an "optimization process."

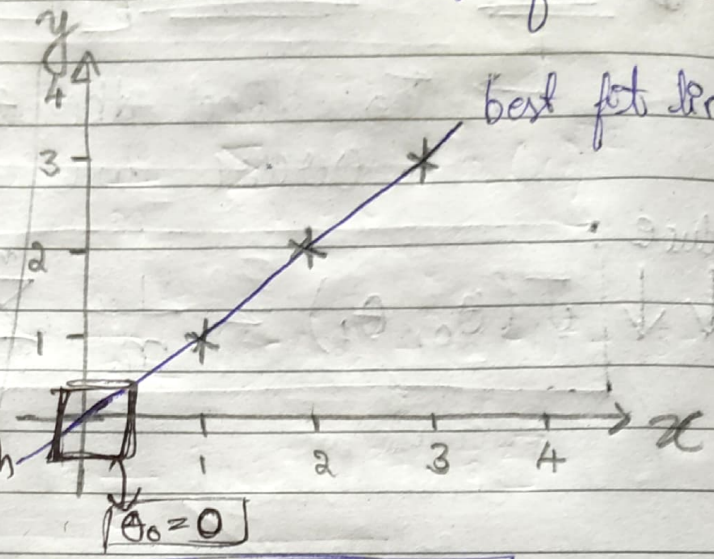
Ex 0

Dataset

x	y	$h_\theta(x)$ ($\theta_1 = 1$)
1	1	1
2	2	2
3	3	3

(independent) (dependent)

Optimization { Minimize the cost function }

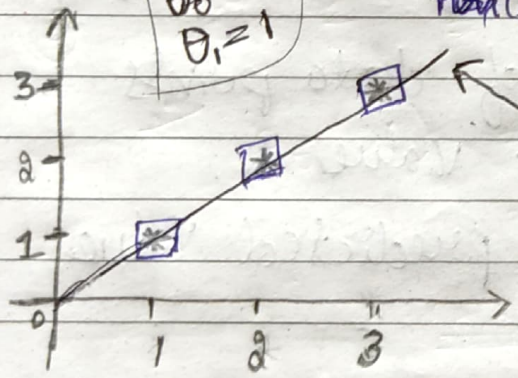


∵ as it touch origin

(1) equation of line : $h_\theta(x) = \theta_0 + \theta_1 x_i$

Ad, $\theta_0 = 0$

hence, $h_\theta(x) = \theta_1 x_i$



let $\theta_1 = 1$

* For $x = 1$

$\Rightarrow h_\theta(x) = 0 + 1(1)$

$h_\theta(x) = 1$

* For $x = 2$

$\Rightarrow h_\theta(x) = 0 + 1[2]$

$h_\theta(x) = 2$

* For $x = 3$

$\Rightarrow h_\theta(x) = 0 + 1[3]$

$h_\theta(x) = 3$

* lets Calculate Cost function

$\theta_0 = 0 \Rightarrow \boxed{h_\theta(x) = \theta_1 x_i}$ $\theta_1 = 1$

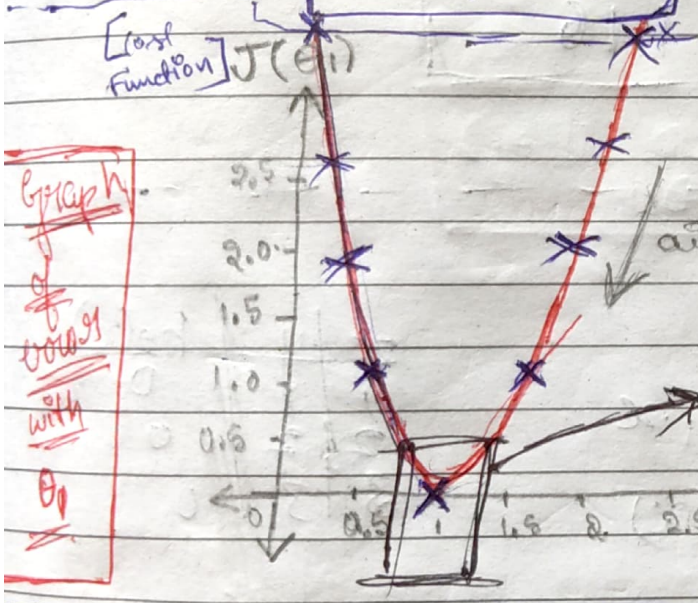
(cost Function) $J(\theta_1) = \frac{1}{n} \sum_{i=1}^{n} [y_i - h_\theta(x_i)]^2$

$y_i - h_\theta(x_i)$ → Error

w.K.T $n = 3$

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

$\boxed{J(\theta_1) = 0}$



Graph of cost with θ_1

⇒ cost Vary with slope

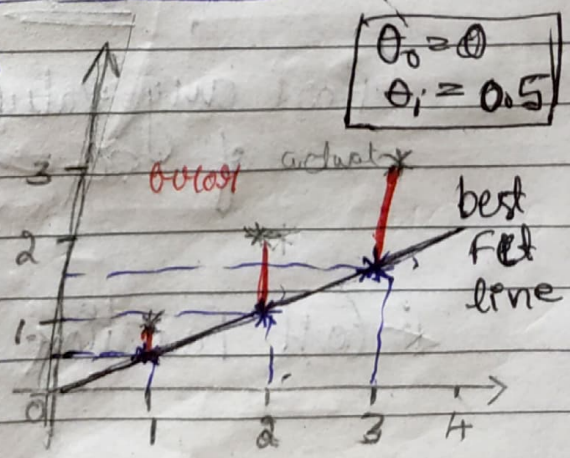
θ_1	$J(\theta_1)$
0	0
0.5	1.16
0	4.66

(2) At $\theta_1 = 0.5$

* $x = 1$, $h_\theta(x) = 0.5(1)$
$= 0.5$

$x = 2$, $h_\theta(x) = 0.5(2)$
$= 1.0$

$x = 3$, $h_\theta(x) = 0.5(3)$
$= 1.5$



let us find error

$\theta_0 = 0$, $\theta_1 = 0.5$

Cost function, $\theta_1 = 0.5$ $\Rightarrow$ $h_\theta(x) = 0.5(x)$

x	y	$h_\theta(x)$ ($\theta_1 = 0.5$)
1	1	0.5
2	2	1
3	3	1.5

$$J(\theta_1) = \frac{1}{n}\sum_{i=1}^{n}\left[y_i - h_\theta(x_i)\right]^2 \quad \text{error}$$

$$= \frac{1}{3}\left[(1-0.5)^2 + (2-1)^2 + (3-1.5)^2\right]$$

$$= \frac{1}{3}\left[0.5^2 + 1^2 + 1.5^2\right]$$

$$\boxed{J(\theta_1) = 1.16}$$

---

(3) let $\theta_1 = 0$, $\theta_0 = 0$

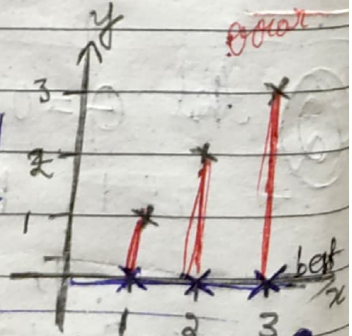
$h_\theta(x) = \theta_0 + \theta_1 x$

$= 0 + 0(x)$

$h_\theta(x) = 0$

x	y	$h_\theta(x)$
1	1	0
2	2	0
3	3	0

For any value of $x = 1, 2, 3$, $h_\theta(x) = 0$



Cost Function, $J(\theta_1) = \frac{1}{m}\sum_{i=1}^{m}\left[y_i - h_\theta(x)\right]^2$

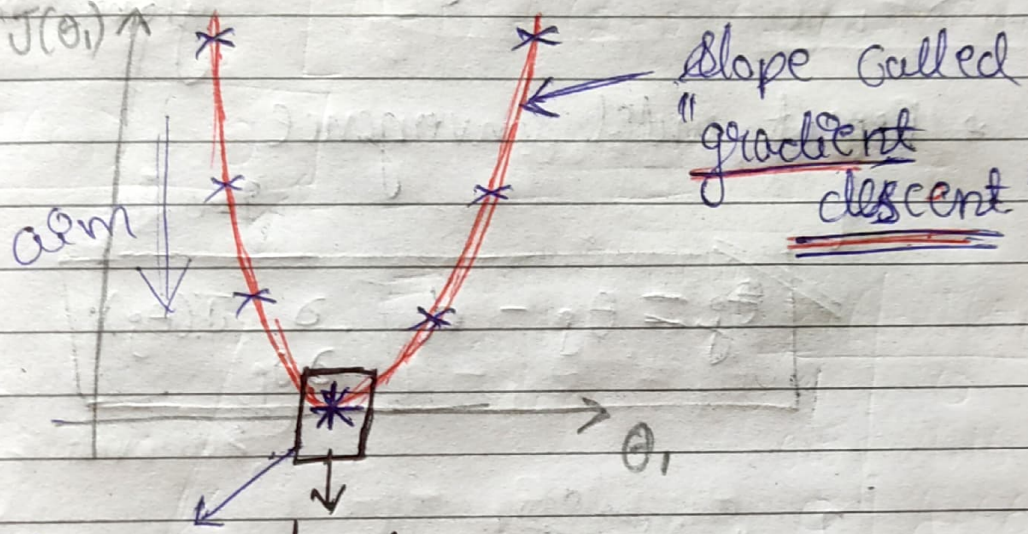
$= 0$

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

$$= \frac{1}{3}\left[1^2 + 2^2 + 3^2\right]$$

$$= \frac{14}{3}$$

$$\boxed{J(\theta_1) = 4.66}$$



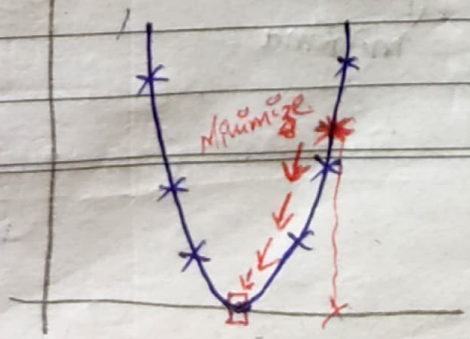
Imp

Slope called "gradient descent"

Error will be least.

global minima
[Cost function ↓↓]

* we can't always initialize a 'θ' value. so we have to find a technique to reduce error.



, we should converge to reach global minima

② Converge algorithm :

→ we need to optimize the change of θ_1

→ we cannot always initialize θ_1 so we have to initialize once then try to converge towards global minima

* Repeat until convergence

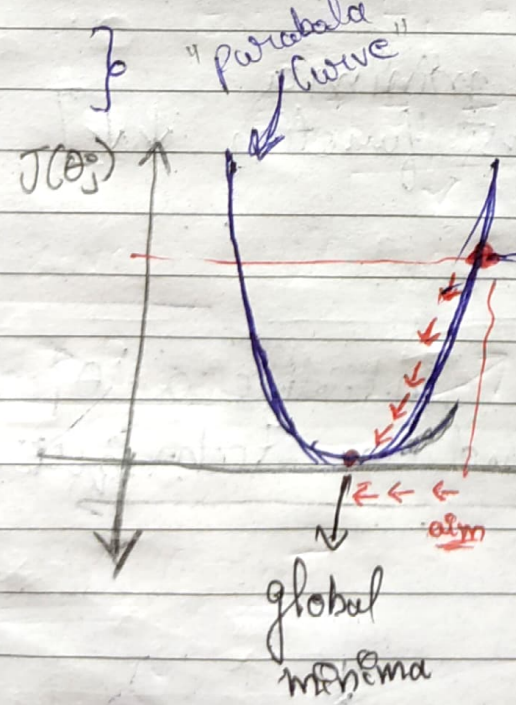
{

$$\theta_j := \theta_j - \alpha \left[\frac{\partial}{\partial \theta_j} J(\theta_j) \right]$$

for $\theta_{0,1}$

→ derivative $= \frac{d}{d\theta_j}[J(\theta_j)]$ means "slope at a specific point".

}



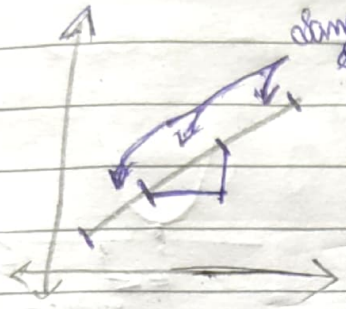
how the formula getting applied?

how to reduce θ_j ?

ho

we need to calculate the derivate of slope to reduce 'θ_j'

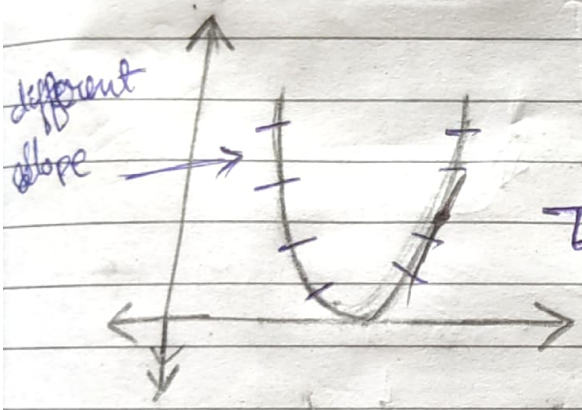
* α = learning rate



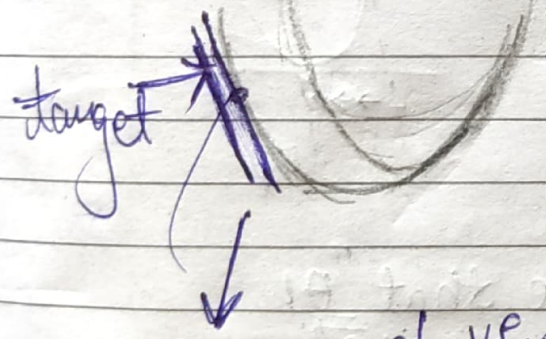
→ For a best fit line we can easily find out the slope of ~~curve~~ line

↓

but



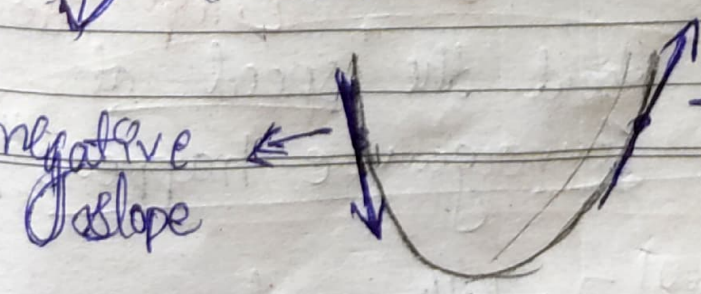
→ In parabola curve.
→ we need derivate of slope of a point.



→ To find slope
→ we have do derivate at point
→ we draw a tangent at the point

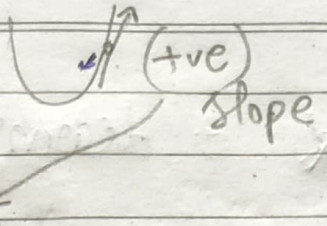
slope can be +ve/-ve.

$$\frac{\partial}{\partial \theta_j} J(\theta_j)$$ → slope at a point



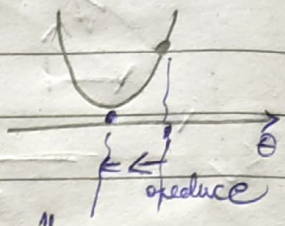
slope

* Now, if slope is +ve



$$\theta_{1\,new} = \theta_{1\,old} - \alpha\,(+ve)$$

$$\theta_{1\,new} = \theta_{1\,old} - (+ve)$$

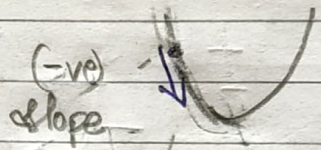


when "slope is +ve"

$\theta_{1\,new} < \theta_{1\,old}$

θ_1 will reduce.

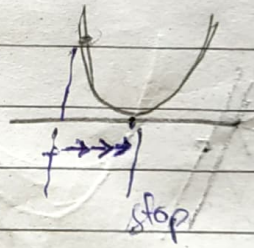
* if slope is -ve



$$\theta_1 = \theta_1 - \alpha\,(-ve)$$

$$\theta_1 = \theta_1 - (-ve)$$

$\theta_{1\,new} = \theta_{1\,old}$ + value



when "slope is -ve"

$\theta_{1\,new} > \theta_{1\,old}$

θ_1 will increase.

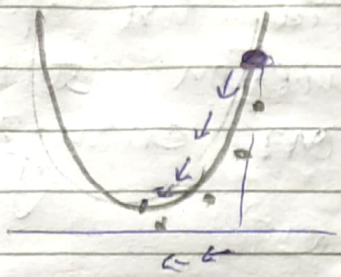
* learning rate

V. V. Imp [Endportant interview]

→ it tells about the "speed at which convergance happens".

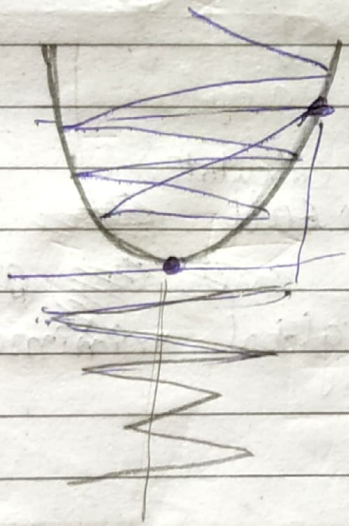
Ex: $\alpha = 0.01$ → tells speed

if $\alpha = 0.01$ [less]



θ, change slowly. reach the global minima.

if α = a [more / high]



θ, change fast and "abruptly move" here and there and near reach the global minima

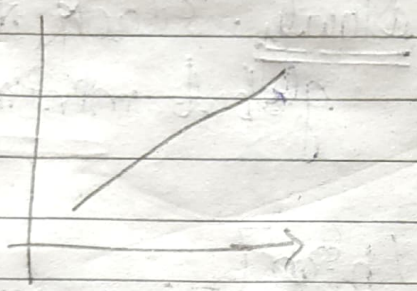
* So, we should choose learning rate (α) that helps in Converge

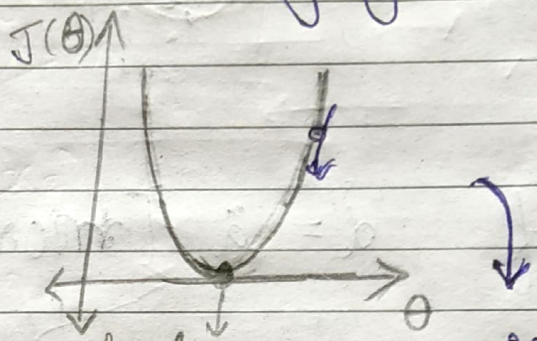
* Conclusion

↳ "we need to repeat Converge algorithm until we reach the global minima where cost function is very less. So, at that point we get best fit line".

* Gradient descent

↳ it is an optimer function that keep on changing derivate





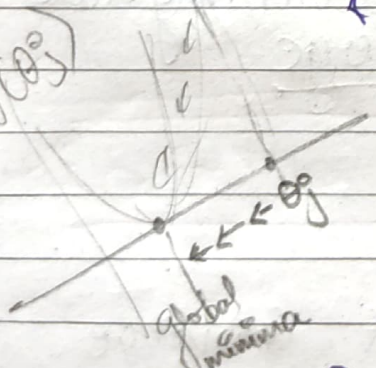
Calculate derivate and again to reach global minima

we have point we try to

* Convergence algorithm

repeat until convergence

{



$$\theta_j^{n} = \theta_j^{o} - \alpha \left[\frac{\partial}{\partial \theta_j} J(\theta_j) \right]$$

}

* Cost function

$$J(\theta_0, \theta_1) = \frac{1}{n} \sum_{i=1}^{n} [y_i - h_\theta(x)]^2$$

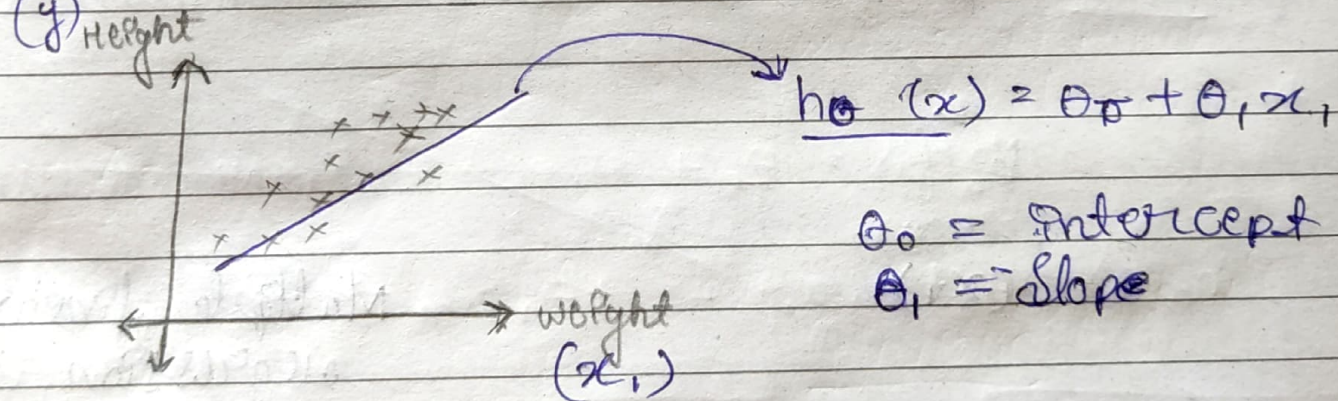
* Multiple linear regression

simple linear

Ex: Dataset

i/p (independent) (x_1) → weight

o/p → Height

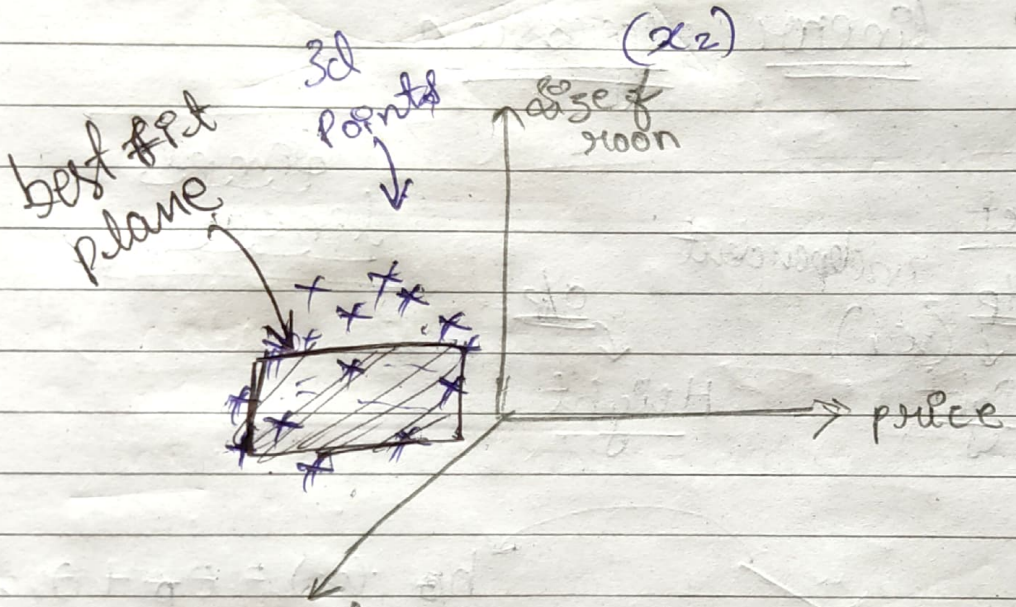


P.T.O

House price Dataset

independent

x_1	x_2	y
No of rooms	size of room	price



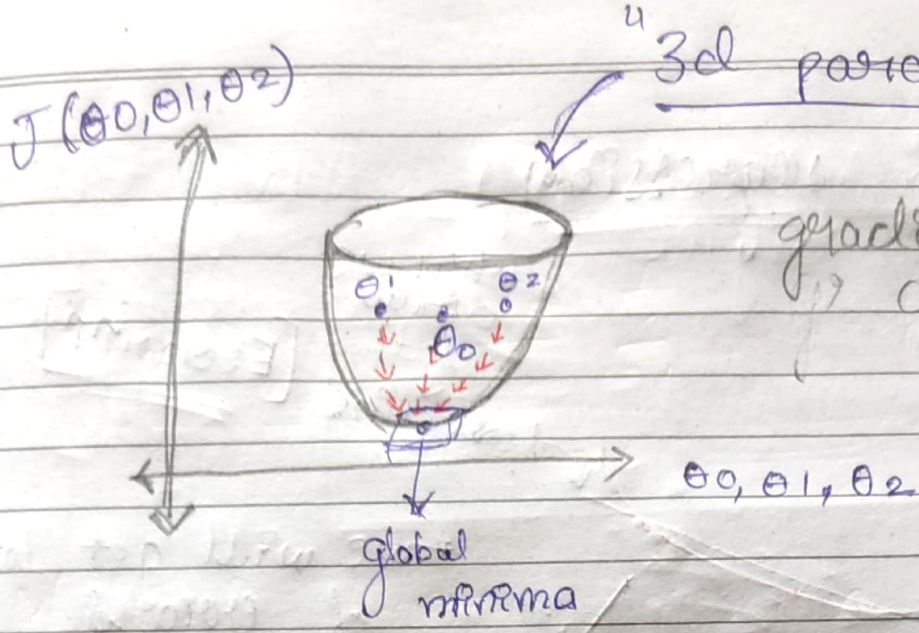
Multiple linear regression

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

$\theta_0 \rightarrow$ intercept

$\theta_1, \theta_2 \rightarrow$ slopes

No of Feature ↑↑ then No of slope ↑↑

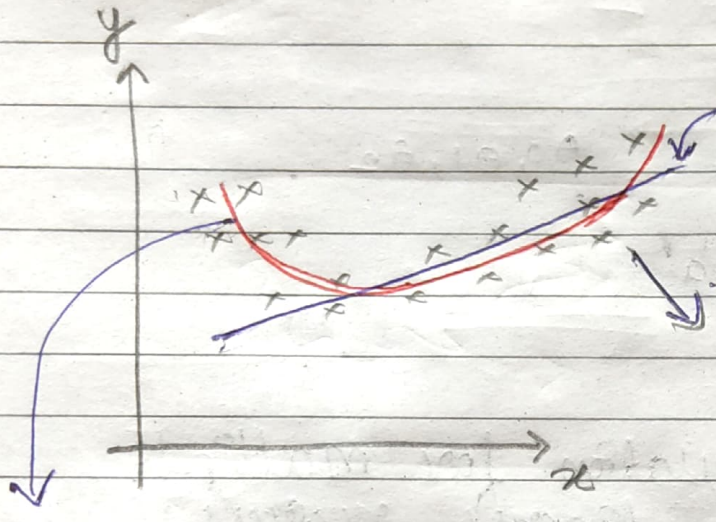


Generic ~~ten~~ equation for multiple linear regression

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

∴ Maximum → Human can see 3dimensional ~~object~~ figures

* Polynomial regression



when we draw fit line

Error

* we will get lot of errors

Non linear relationship data.

we have made a best fit curved line. for a non-linear relationship data.

*

$h_\theta(x) = \theta_0 + \theta_1 x$ → linear regression

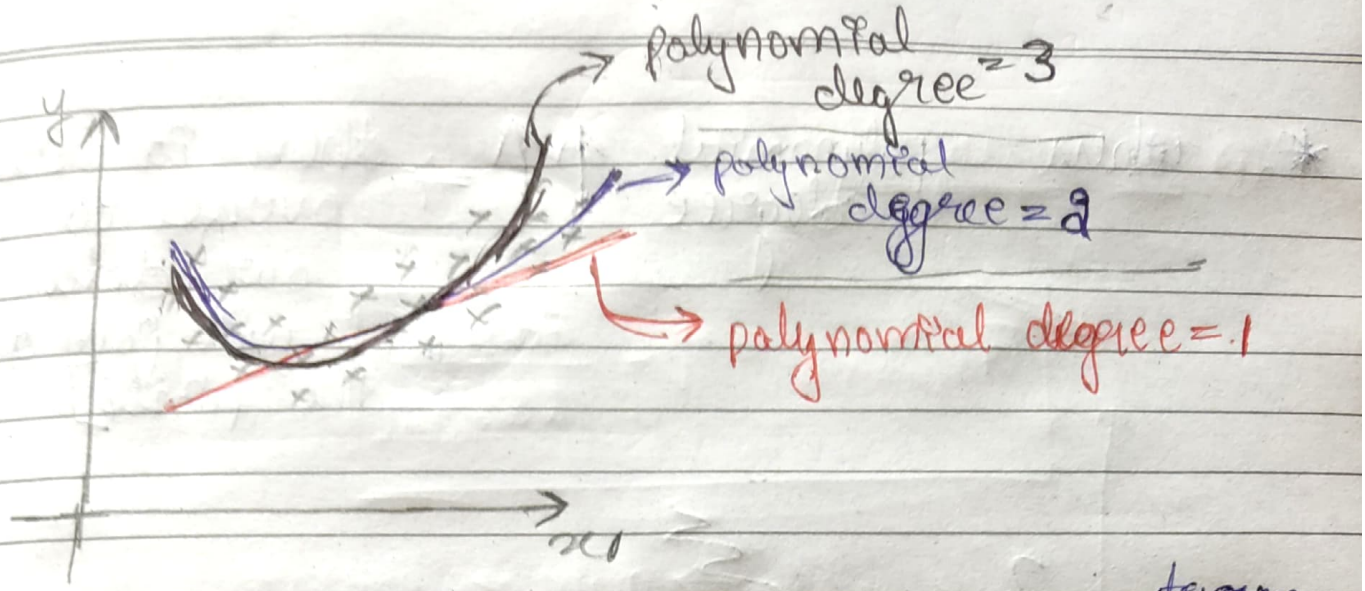
$h_\theta(x) = \theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n$ → multiple regression

← Polynomial regression

used when data is having non-linear relationship.

↓

Polynomial degrees.



* polynomial degree = 0 $\Rightarrow$ $h_\theta(x) = \theta_0 + \underbrace{\theta_1 x_1}_{0}$ ~~term~~ becom

$h_\theta(x) = \theta_0 * 1$

$h_\theta(x) = \theta_0 * x_1^{(0)}$

* polynomial degree = 1 $\Rightarrow$ $h_\theta(x) = \theta_0 * x_1^{(0)} + \theta_1 * x_1^{(1)}$

[simple linear regression]

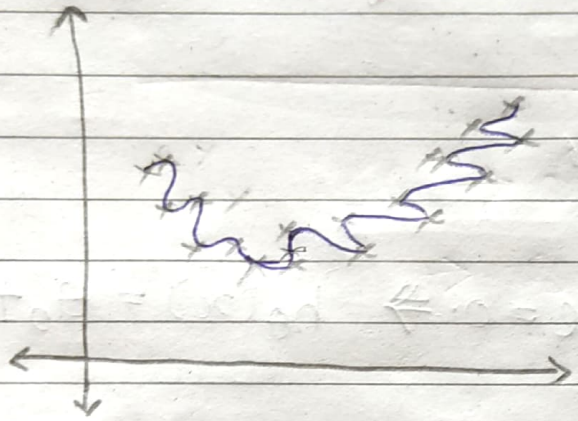
* polynomial degree = 2 $\Rightarrow$ $h_\theta(x) = \theta_0 x_1^{(0)} + \theta_1 x_1^{(1)} + \theta_2 x_1^{(2)}$

Curve fit eyes better for our data

* polynomial degree = 3,

$h_\theta(x) = \theta_0 x_1^{(0)} + \theta_1 x_1^{(1)} + \theta_2 x_1^{(2)} + \theta_3 x_1^{(3)}$

* when polynomial degree ↑↑ ⇒ then it will overfit to our data.
[it will join all the points]



∴ we need to find a suitable polynomial degree. So, that it donot overfit to data.

* polynomial degree = n

$$h_\theta(x) = \theta_0 x_i^0 + \theta_1 x_i^1 + \theta_2 x_i^2 + \dots + \theta_n x_n^n$$

⇓

Simple polynomial regression
[just for 1 independent feature & 1 dependent feature]

* Multiple polynomial Regression

{ Multiple independent features }

independent	dependent
x_1 x_2	y

then our equation for polynomial degree = 2

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2$$

* Performance metrics used in linear regression

- * What is the accuracy?
- * How do we calculate it?
- * when do we say that we got better model/not

We have 2 different metrics

1. R squared.
2. Adjst R squared.

Interview

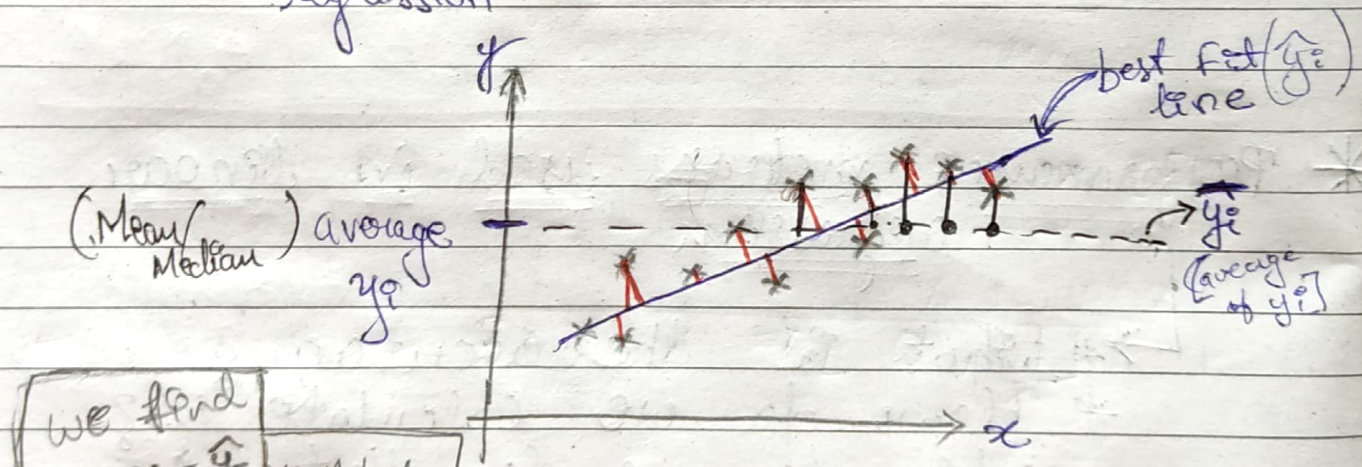
① R Squared:

Any error getting from best fit line should be less than error getting from average line.

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

* SS → Sum of square
Res → residual [difference of y & $\hat{y}$]
(or) error

Aim of Simple linear regression ⇒ To get Best Fit line



we find $y - \hat{y}$ from best fit line

* SS_{Res} ⇒ Sum of square residuals / error

SS_{Total} ⇒ Sum of square total.

[difference b/w actual point & predicted point w.r.t to mean line]

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

error w.r.t best fit line

small value / Big value

error w.r.t average line of y_i

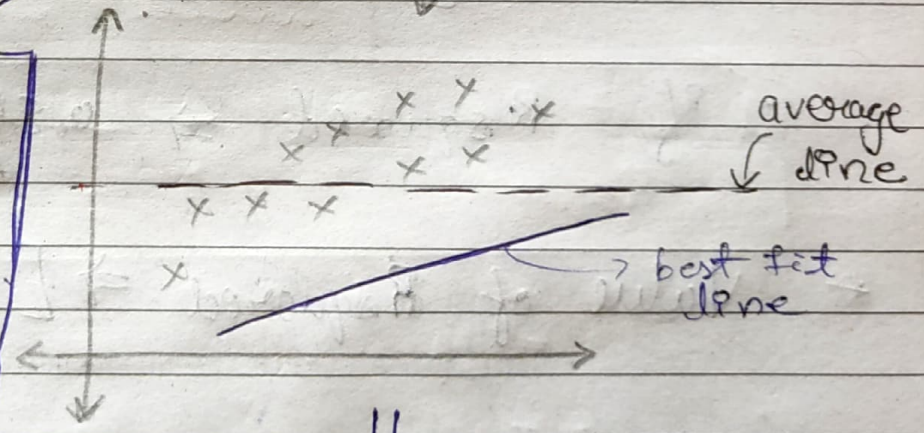
small the value better the accuracy.

Interview asked

Q) Can "R squared have -ve value" ?

⇒ Yes, only if the best fit line is worse.

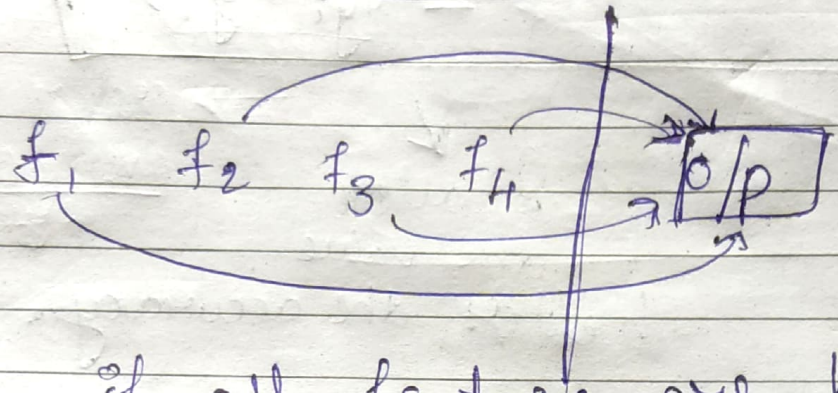
error in best fit line > error in average line.



⇓

$R^2_{squared}$ will be "-ve"

* Suppose ⇓



if all feature are highly correlated with output value then ⇓

↑↑ $R_{squared}$ will increase.
[better accuracy]

* if $R_{squared} \Rightarrow 0.85$ then 85%.

* Value of $R_{squared} \Rightarrow [0, 1]$ (0% to 100%)

Maximum value.

* we should not get $R_{squared} = 1$ because it tell model is "overfitted"

② Adjusted R squared:

Eg:

Independent		dependent
Size of house	No of bed rooms	Price
x_1	x_2	y

Here, x_1 & x_2 are directly Correlated with price.

suppose → R squared = 85%

* Now, if we had "location" ~~feature~~ (x_3)

it is also highly correlated with output

~~but~~ → R squared = 90% drastically increase

* Now, if we add "gender"

less Correlation ↓

→ R squared = 91% less increase

* So, when we add on feature our R square will keep on increasing.

"R square" → "tells how model is behaving with addition of new feature"

This will not happen with "adjusted R square".

To prevent this we use.

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

N = No of data points

P = No of independent features

Ex: Suppose, $R^2 = 0.8 \Rightarrow 80\%$, $N = 11$, $P = 2$

$$\text{adjusted } R^2 = 1 - \frac{(0.2)(10)}{11-2-1} = \frac{1-2}{8}$$

$$= 1 - 1/4$$

$$= 3/4 = 0.75$$

when. $R^2 = 80\%$ then → adjusted $R^2 = 75\%$

$R^2 > \text{adjusted } R^2$ (always)

* if we add 1 (v.imp) independent feature.

$\uparrow\uparrow R^2 = 85\%$ → adjusted $R^2 = 78\% \uparrow\uparrow$

* if we add 1 independent feature (not imp)

$\uparrow R^2 = 86\%$ → adjusted $R^2 = 76\% \downarrow$

better unit to measure accuracy

*	R-Square	Adjusted R-Square
* For any new imp feature added	increase drastically ↑↑	increase ↑↑
* not imp feature added	increase by small amount ↑	decrease by small amount ↓

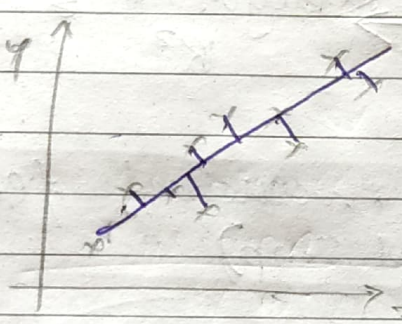
* MSE, MAE and RMSE

[Cost function]

↳ Used to quantify how model prediction align with actual values.

Explained clearly

* To reduce error we consider



"Error" should be minimum

↓

To measure error we use [cost function].

we say

$$J(\theta_0, \theta_1) = \frac{1}{n} \sum_{i=1}^{n} [y_i - h_\theta(x_i)]^2$$

↓

This "mean - squared error"

↑

Cost function is called as

Our aim is to reduce this error

↓

We reduce this error by using → "Convergence algorithm"

① Mean Squared error (MSE):

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

→ Quadratic equation

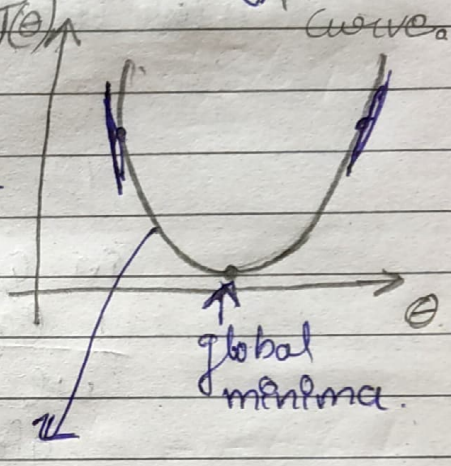
⇓ we get parabola curve.

we can have

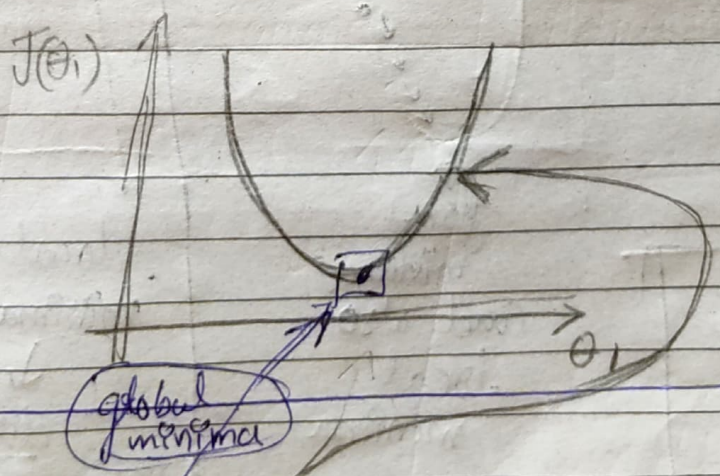
$ax^2+bx+c=0$

$(a-b)^2 = a^2+b^2-2ab$

we reduce error using "convergence algorithm"



this curve is called "Convex function"



Until we reach optimal solution by ~~using gradient~~ by using 'gradient descent'

→ used to minimize (or) maximize the function

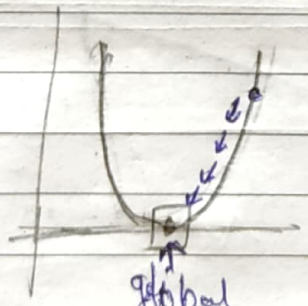
* Advantages

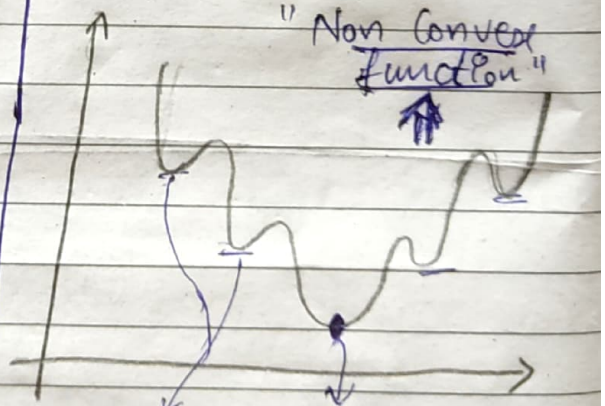
① It is differentiable.

i.e at any point we are able find slope of the point.

② It has one local and one global minima.

In quadratic Equation	In Non-quadratic equation
we know our global minima we know where we need to converge	In graph we have many local minima





And also local minima

So, we can't move

value donot change

slope=0

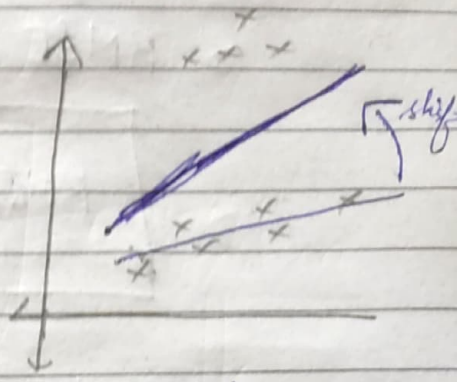
disadvantage

It will stuck on that region as it cannot calculate slope.

Convergence $\Rightarrow \theta_0 = \theta_0 - \alpha\left[\frac{\delta}{\delta\theta_0} h_\theta(x)\right]$

* Disadvantages

① Not Robust to outliers.
best fit line shift more from actual position.



② It is not in the same unit.

Price

INR → $(y_i - \hat{y})^2$ → $(INR)^2$

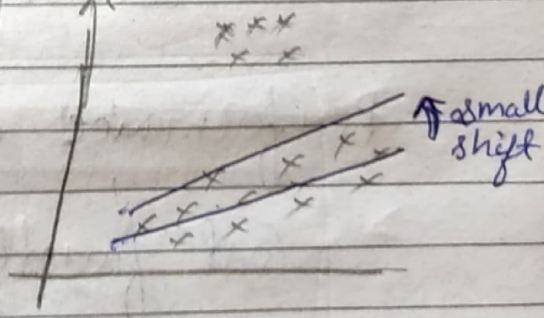
Price

---

② Mean absolute error. [MAE]

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

→ only magnitude



Advantage

① Robust to outlier.

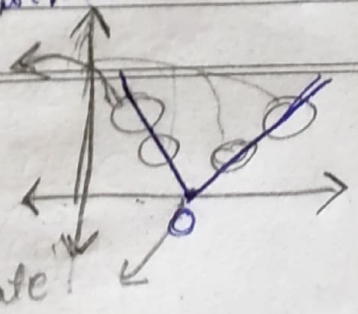
② It will be in same unit.

Disadvantage

① Convergence usually takes time.
we calculate "sub gradient"

"optimization is Complex"

② Time



(3) RMSE [Root mean Square error]

$$RMSE = \sqrt{MSE}$$

$$RMSE = \sqrt{\frac{\sum (y_i - \hat{y}_i)^2}{n}}$$

* Advantages
  1. Same unit.
  2. Differentiable.

* Disadvantage
  1. Not robust to outlier.

Note:
Linear regression.

Performance check → R^2 and adjusted R^2

Cost function → MSE, MAE, RMSE

* Simple linear regression in python

> plt.scatter(. —, —)

① Dataset division (independent & dependent)
② Train, test split of dataset.
③ Standardization.
④ Model training ⑤ testing & final accuracy.

> from sklearn.model_selection import train_test_split

> x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, 20% random_state = 42)

> from sklearn.preprocessing import StandardScaler

> scaler = StandardScaler()

> { scaler.fit_transform(x_train)
  scaler.transform(x_test)

train model > from sklearn.linear_model import LinearRegression

> regressor.fit(x_train, y_train)
finds θ_0 & θ_1

> regressor.coef_ ← give slope

> regressor.intercept_ ← intercept

> plt.plot(x_train, regressor.predict(x_train))

> y_pred = regressor.predict(x_test)
test data

plt.plot(x_test, y_pred)
predicted data

Performance metrics

> from sklearn.metrics import mean_squared_error, mean_absolute_error

> mse = mean_squared_error(y_test, y_pred)
actual predicted

> rmse = np.sqrt(mse)

> from sklearn.metrics import r2_score

Accuracy = r2score(y_test, y_pred)

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

$adj = 1 - \frac{(1-r^2)(n-1)}{(n-p-1)}$

* Multiple regression with python

> from sklearn.datasets import fetch_california_housing

> california = fetch_california_housing()

> from sklearn.model_selection import train_test_split

(2) split

> x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.33, random_state = 10)

(3) standardize

> from sklearn.preprocessing import StandardScaler

(4) > from sklearn.linear_model import LinearRegression.

> regressor.fit(x_train, y_train)

train

> regressor.coef_

> regressor.intercept_

(5) predict

y_pred = regressor.predict(x_test)

⑥ error

> from sklearn.metrics import mean_squared_error
mean_absoalte_error

mse_error = mean_squared_error(ytest, y_pred)

---

⑦

accuracy > from sklearn.metrics import r2_score

r_square = r2_score(y_test, y_pred)

print (r_square * 100)

$$\%\,acc_y = 1 - \frac{(1-r^2)(n-1)}{(n-1-p)}$$

---

⑧ data

new data

scaled_data = scaler.fit_transform(data)

pred = regressor.predict(scaleddata)

---

⑨ Assumptions

① scatterplot (ytest, ypred) → linear graph

② dist plot (residuals) → normal distribution

③ scatter plot (y_pred, residuals) → uniform distribution

(10) Pickling

> import pickle

> pickle.dump(regression, open("model.pkl", "wb"))
file mode

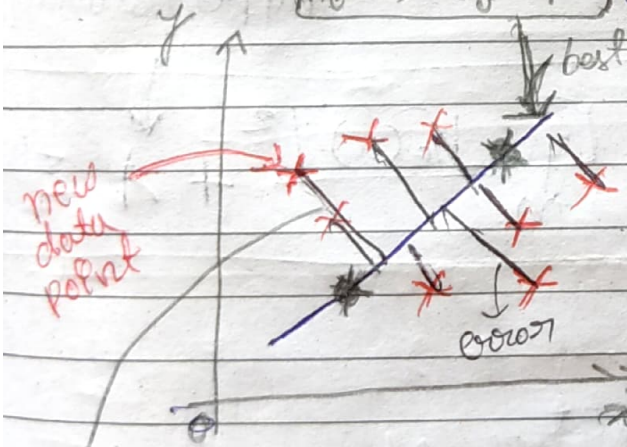
> model = pickle.load(open("model.pkl", "rb"))

> model.predict(new_scaled_data)

* Ridge regression [L2 regularization]

used to "reducing overfitting"

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



Dataset

train → Train our model

test → Test our model

$\therefore$ Cost function = $\frac{1}{n}\sum_{i}^{n}[y_i - h_\theta(x_i)]^2$

$n \Rightarrow$ 0 ← as error is 0 i.e model is trained properly.

Error is more, i.e accuracy decreases ⇒ This is called overfitting [low bias, high variance]

* overfitting

Train dataset → Accuracy ↑↑

Test dataset → Accuracy ↓↓

Training testing

low bias and High variance.

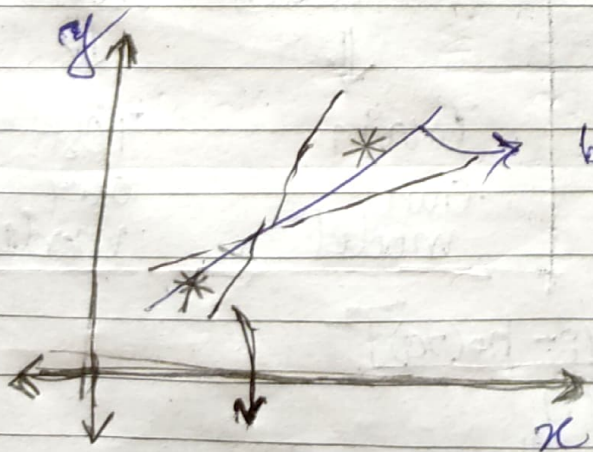
↓ error is more as data spread is more. So, accuracy is less.

* In order to overcome the above problem we use ridge regression.

* Ridge regression

hyperparameter

cost function $J(\theta) = \frac{\sum_{i=1}^{n} [y_i - h_\theta(x)]^2}{n} + \lambda \Sigma$



best fit line should not pass through 2 points because error will be 0.

best fit line may be rotated like this.

* let $\lambda = 1$

$$h_\theta(x) = \theta_0 + \theta_1 x_1$$

$\theta_1 \rightarrow$ slope

then

$$\text{Cost function} = \frac{\sum [y_i - h_\theta(x_i)]^2}{n} + \lambda \sum (\text{slope})^2$$

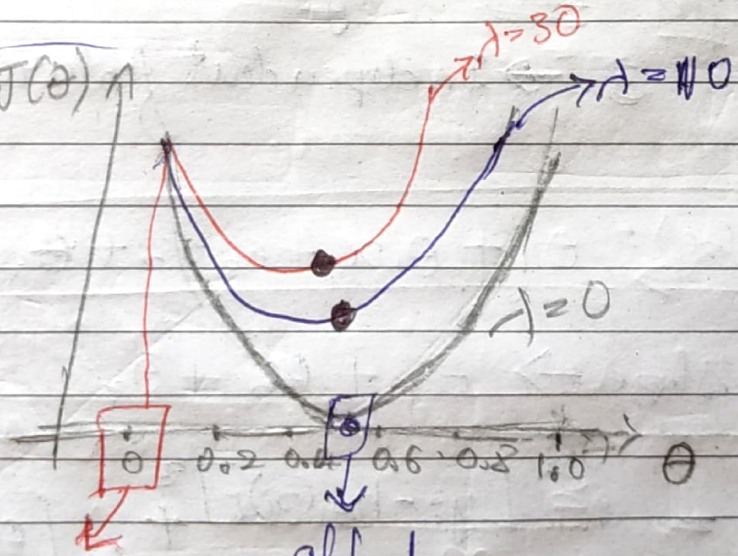
[by penalizing cost function] $= 0 + \lambda [(\theta_1)^2]$ ⇒ penalizing cost function

↳ error will not be 0.

reduce overfitting

Always ∴ cost function > 0 and never 0.

*



gradient descent

$(\text{slope}\theta)^2$

never become 0.

global minima

* As $\lambda \uparrow\uparrow\uparrow$ → global minima move upwards.

↳ slope ↓↓

* Simple linear regression

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

$$\text{Cost function} = \frac{\sum_{i=1}^{n} [y_i - h_\theta(x)_i]^2}{n} + \lambda \left(\sum_{i=1}^{n} (\text{slope})^2 \right)$$

$$[\theta_1^2]$$

d2 norm

* Multiple linear regression

3 ⇒ Independent feature [D]

$$h_\theta(x) = \theta_1 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$

slope ⇒ $\theta_1 + \theta_2 + \theta_3$

$$\text{Cost function} = \frac{\sum_{i=1}^{n} [y_i - h_\theta(x)_i]^2}{n} + \lambda [\theta_1^2 + \theta_2^2 + \theta_3^2]$$

* Lasso Regression [L1 regularization]

↓

"Feature selection"

$$\text{Cost function} = \frac{1}{n}\sum[y_i - h(\theta x)]^2 + \boxed{\lambda \sum_{i=1}^{n} |\text{slope}|}$$

* Lasso, Consider

MLR, $h_\theta(x) = \theta_1 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$

tells that unit change in 'x' how much change in 'y'

$= 0.52 + 0.65 x_1 + 0.72 x_2 + 0.12 x_3$

tells that unit change in 'x_3' there is only 0.12 change in 'y'

we can conclude that x_3 has less correlation with output feature (y)

so, we can remove this 'x_3' feature.

---

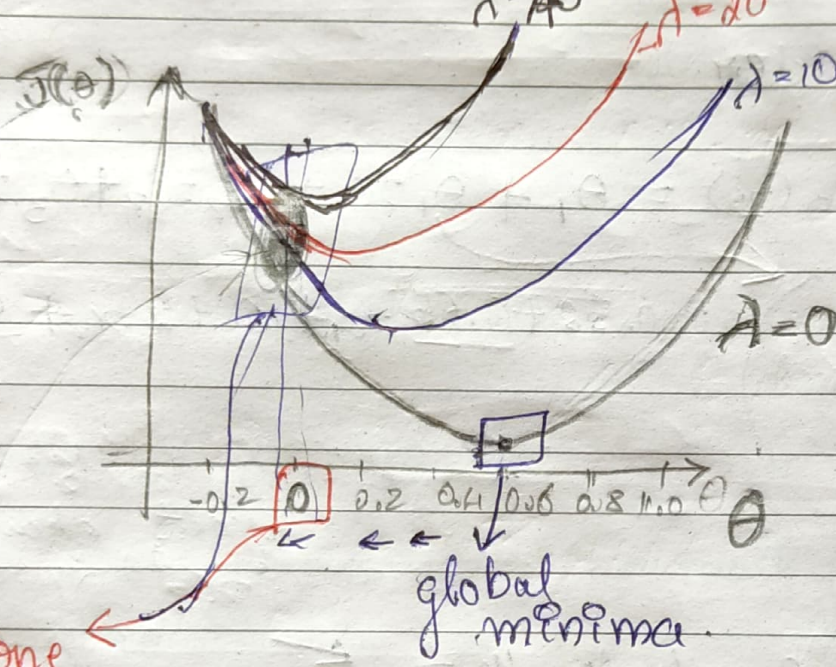
* For the above process we use lasso regression.

Cost function = $\frac{\sum [y_i - h_\theta(x_i)]^2}{n} + \lambda \sum_{i=1}^{n} |\text{slope}|$

⇓

if $\lambda = 1$,

Error + $1\,[|\theta_1| + |\theta_2| + |\theta_3|]$



gradient descent

at one point it become 0. → but in ridge regression it will never become 0

↓ which

'θ' is less correlated become 0.

* Now,

$h_\theta(x) = 0.52 + 0.65x_1 + 0.78x_2 + 0.12x_3$

0

This less correlated feature will actually become 0 at 1 point.

Here, feature selection has happened

$\lambda \uparrow\uparrow\uparrow$ slope $\downarrow\downarrow$

At one point, $\boxed{\theta \approx 0}$

* Elastic Net regression → Lasso + Ridge

Combination of

① Reduce overfitting [Ridge] | ② Feature selection [lasso]

$$\text{Cost function} = \frac{\sum_{i=1}^{n} [y_i - h_\theta(x)_i]^2}{n} + \lambda_1 \sum_{i=1}^{n} (\text{slope})^2 + \lambda_2 \sum |\text{slope}|$$

MSE | Reduce overfitting [L2 regularization] | Feature selection [L1 regularization]

$\lambda_1, \lambda_2 \Rightarrow$ { hyperparameter tuning }

If ratio = 0.5 remaining given to L2 regularization

* Ridge, lasso, ElasticNet with python

Ridge → from sklearn.linear_model import Ridge

lasso → from sklearn.linear_model import Lasso

elasticnet → from sklearn.linear_model import ElasticNet

* All the supervised learning model have ↓

fit and predict method

→ regressor.fit(x_train, y_train)

→ regressor.predict(x_test)

* Algerian forest

① # cleaning

> df.loc[:122, 'Region'] = 1 — New feature
> df.loc[122:, 'Region'] = 2.
(122: row, Region: column)

> df = df.dropna().reset_index(drop=True)

> df.iloc[[122]]

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

>

> df[[-,-,-]] = df[[-,-,-]].astype(float)

>

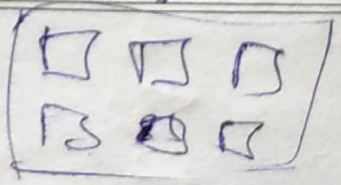
② EDA

> df['class'] = np.where(df['class'].str.contains('fire'), 1, 0)

> drop → day, month, year.

— plot

> plt.style.use('seaborn')
df.hist(bins=50, figsize=(20,15))
plt.show()
—



> percent = df_copy['classes'].value_counts(normalize)*100

plt.pie(percent, labels = class_labels, autopct='%1.1f%%')

[FP, INF]

Training

① Divide dependent and independent feature.

② Train-Test split.

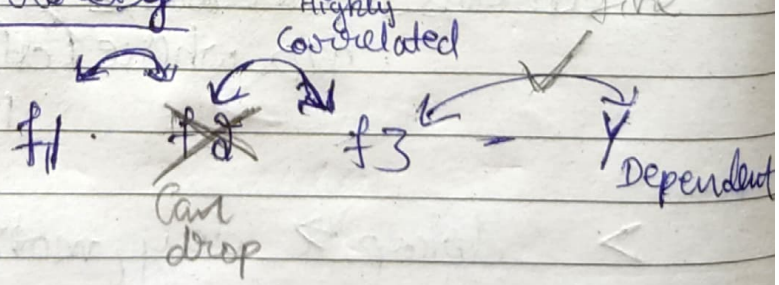
③ Feature selection.

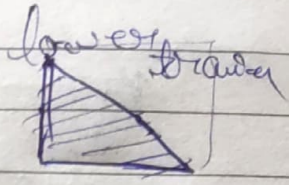
↳ > 85% correlation drop.

domain expert gives threshold

because same o/p not needed [similar feature should be dropped]

Multicollinearity






```
> def correlat(df, threshold):
      col_corr = set()
      col_matrix = df.corr()
      for i in range(len(df.columns)):
          for j in range(i)
              if abs(corr_matrix.iloc[i,j]) > thres):
                  col_name = corr_matrix.columns[i]
                  col_corr.add(col_name)
      return col_corr
```

-ve value

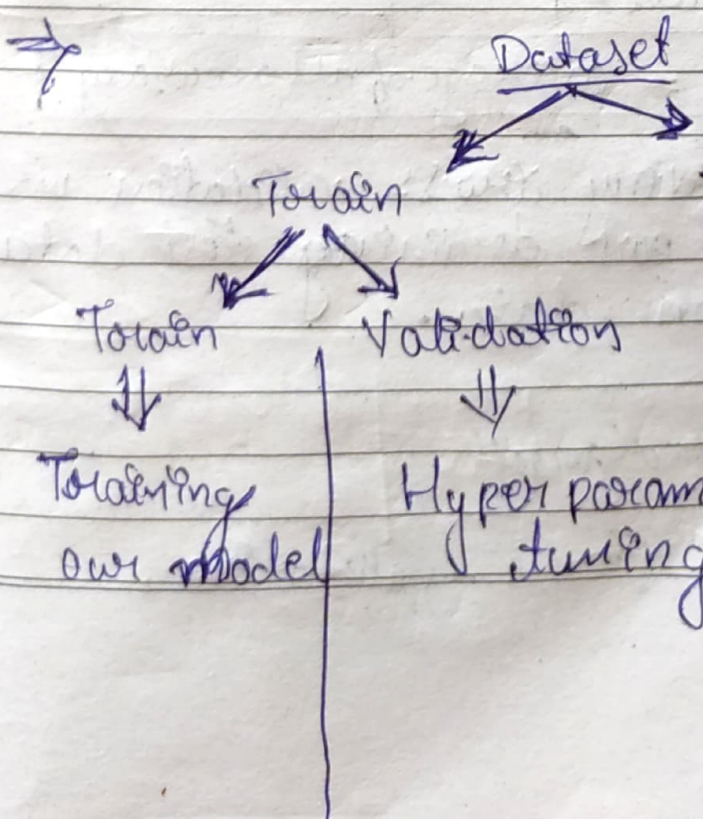
> x_train, x_test → drop col_corr.

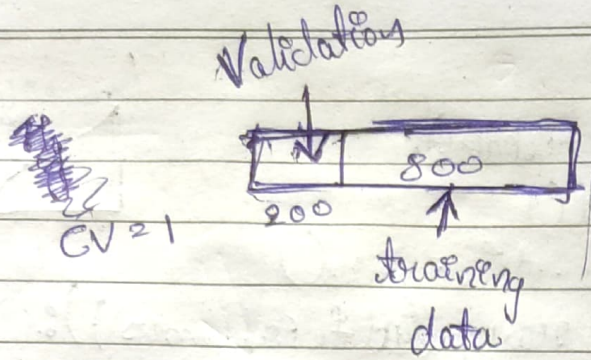
(4) Standardize (x_train, x_test)
understand imp of scaling

(5.1) linear regression
> from linear_model import LinearRegression

(5.2) lasso regression (Feature selection)
> import Lasso

(5.3) cross validation Lasso.



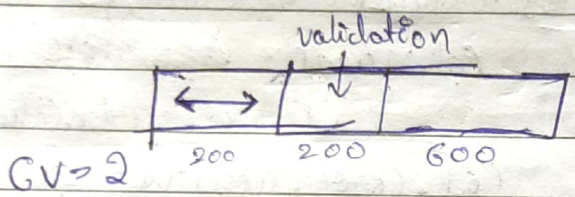


Training
1000 data points

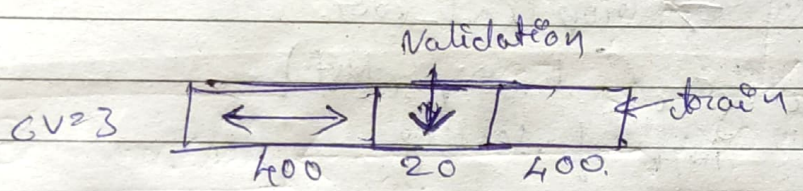
Cross validation = 5

$\frac{1000 \text{ data}}{5} = 200$

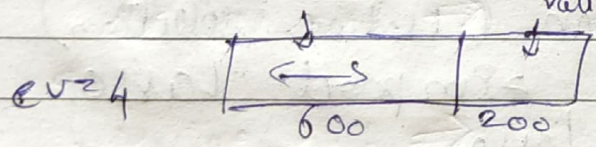
$\Rightarrow$ Acc = 1



$\Rightarrow$ Acc = 2



$\Rightarrow$ acc = 3



$\Rightarrow$ acc = 4

Validation data

↓

used for hyperparameter tuning.

* Final accuracy = average of all accuracy

Here, in every cross validation we change our training data & validation data.

↓

To check model

So, this called Cross-validation.

* > from sklearn.linear_model import LassoCV

lasso_cv = LassoCV(cv=5)

rest all same

5.4 Ridge regression - reduce overfitting

> from sklearn.linear_model import Ridge

5.5 RidgeCV - cross validation

> import RidgeCV

5.6 Elastic net regression

> import ElasticNet

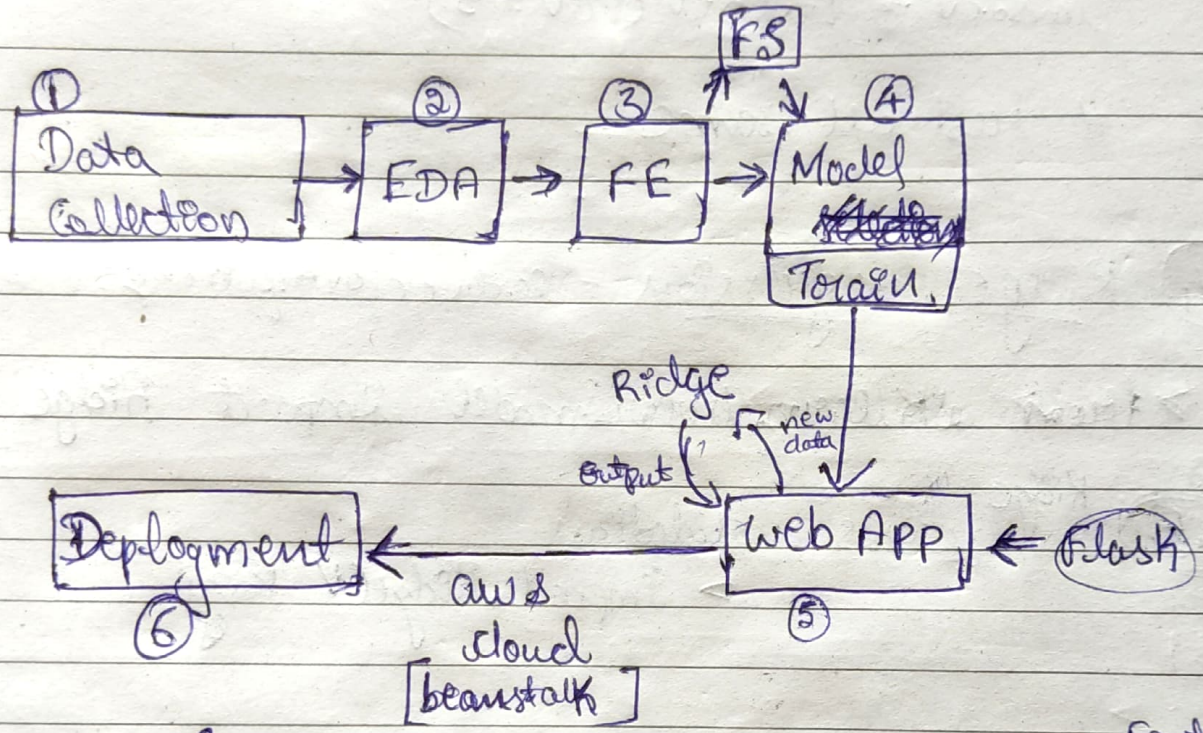
5.7

> import ElasticNetCV

6

* Machine learning Projects

ML Project life cycle



{ Scaler.pkl → Standardization → Feature Scaling
  Ridge.pkl → Data Prediction → Model

---

* Folders

1. notebooks → Dataset, EDA, model train.ipynb } For reference
2. models → scaler.pkl, ridge.pkl
   4. requirement.txt
   5. application.py
3. templates
   - index.html
   - home.html

add all the library required for project

> pip install -r requirements.txt

It will install all library to environment when run command.

* to automatically add required libr
> python -m pipreqs.pipreqs

application.py

> from flask import Flask, request, jsonify, render_template

> import pickle

> import numpy as np
pandas

> from sklearn.preprocessing import Standard Scaler

> ridge_model = pickle.load(open('model/ridge.pkl','rb'))
std_scaler = ridge.pkl scaler.pkl

> app = Flask(__name__)

@app.route('/')
def index():
return render_template('index.html')

goes to template folder and renders 'index.html' file

* GET ⇒ getting a static page.

POST ⇒ requesting posting some info and getting response to that particular info.

```
@app.route("/predictdata", method=["GET", "POST"])
def predict_datapoint():
    if request.method == "POST":
        temp = float(request.form.get("Temp"))

        new_data = [ ____ ____ ]
        new_scaled_data = scaler.transform([new_data])
        result = ridge_model.predict(new_scaled_data)

        return render_template("home.html", result=result[0])
    else:
        return render_template("home.html")

if __name__ == "__main__":
    app.run(host="0.0.0.0")
```

---

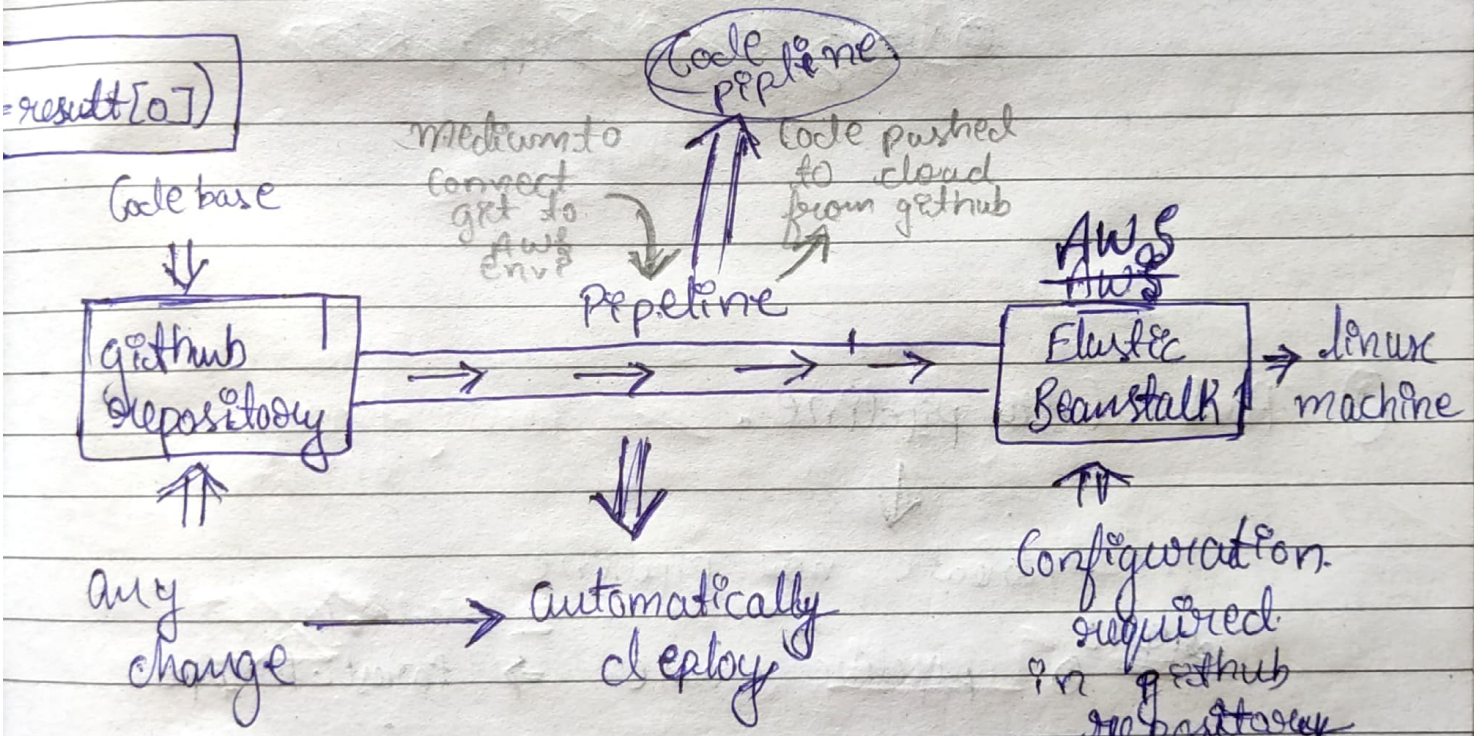
(6) home.html

```
<form action="{{ url_for('predict_datapoint') }}"
      method="post">
    <input type="text", name="Temp",
           placeholder="__" required="required" />
    <button type="submit"> Predict </button>

</form>

<h2> {{ results }} </h2>
```

→ home.html

```
<input type="text" name="Temp" />
```

AWS deployment



Folders → + add ① .ebextensions
↳ python.config

"/predict data"

In AWS

① AWS Elastic Beanstalk.

↓ application

Create application

step 1: ① Application name ⇒ github repo name

② Platform ⇒ Python

③ select sample application

step 2: ① Source provider ⇒ github (select version)

② Connect github.

③ select repo.

④ select branch.

step 3: skip (for now)

step 4: select Beanstalk.

+ name of environment

② AWS Code pipeline.

↓

Create new pipeline

step 1: ① pipeline name ⇒ forest-fire-pipe

step 2: ① Source provider ⇒ github (select version)

② Connect github.

③ select repo.

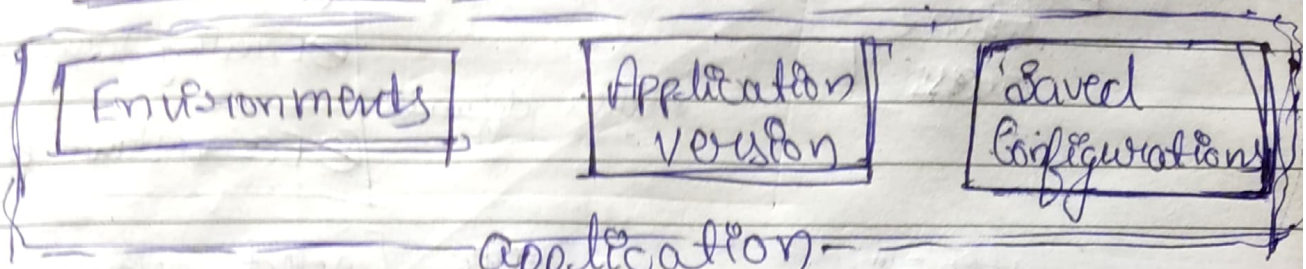
④ select branch.

step 3: skip (for now)

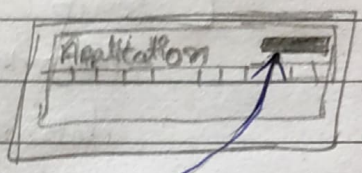
step 4: select Beanstalk.

+ name of environment

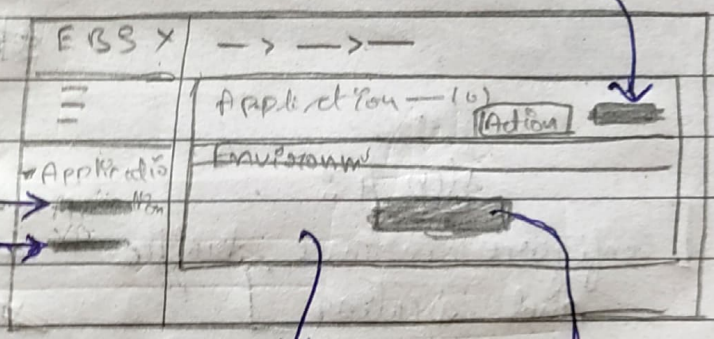
→ Elastic Beanstalk.



① Create application [Folder]
- enter name
- ② ~~Create~~ new application



② * we can see.



② Create environment

step 1: [Configure environment]
- * web server environment
- * Application Name [already ~~Created~~]
- * platform → python [select]
- * Sample application [selected]

For first time we have to create it

step 2:
[Configure service access]

* Use an existing service role.
  1. existing service role
  2. EC2 instance profile
* other default

step 3:
[Setup networking, database & tags]

* VPC [select default]
* instance settings
* select 2 different subnets
* database
* defaults

step 4:
[Configure instance traffic and scaling]

* default
* Security group

step 5:

* Monitoring
* System [select Basic]
  Enhanced
* Managed platform updates
  * deactivate
* defaults

step 6: Submit

Need to create IAM role

AWS Console → IAM

↓

Roles

① Create role

step 1: ② - AWS service.

③ Use case → select Elastic Beanstalk

step 2: ④ policy

↓ → ~~AWS Elastic Beanstalk Empty~~ Web Tier

~~step 3~~ imp "AWS Elastic Beanstalk Web Tier"

step 3:

⑤ Role name

Submit

* Logistic Regression

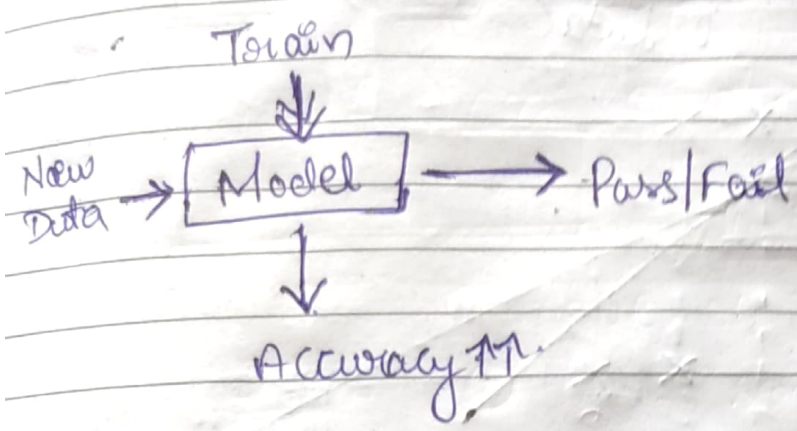
→ Used to solve classification problem

- Binary classification → o/p : 2 categories
- Multi class classification → o/p : > 2 categories

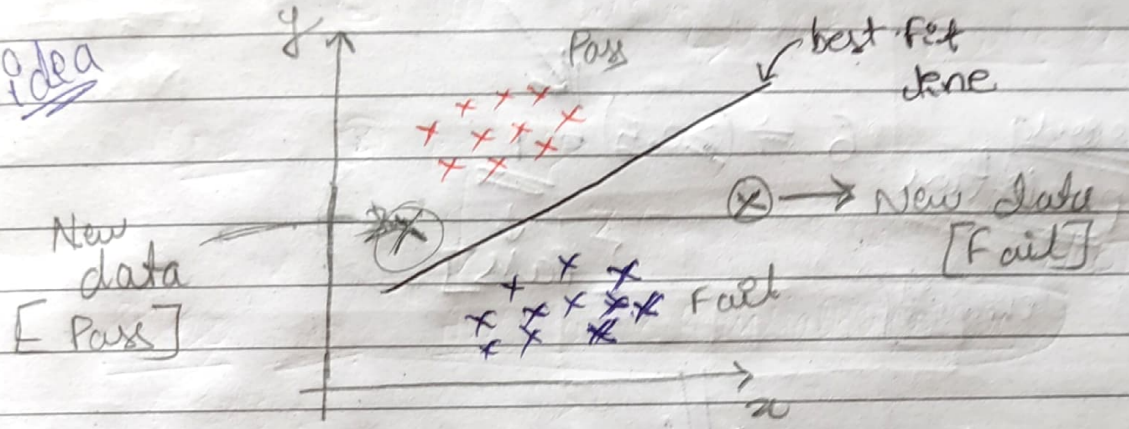
* Consider,

Dataset

No of play hours [Independent] i/p	Pass / Fail [y] o/p [dependent feature]
9	Fail [0]
8	Fail 0
7	Fail 0
6	Fail 0
5	Pass [1]
4	Pass 1
3	Pass 1
2	Pass 1

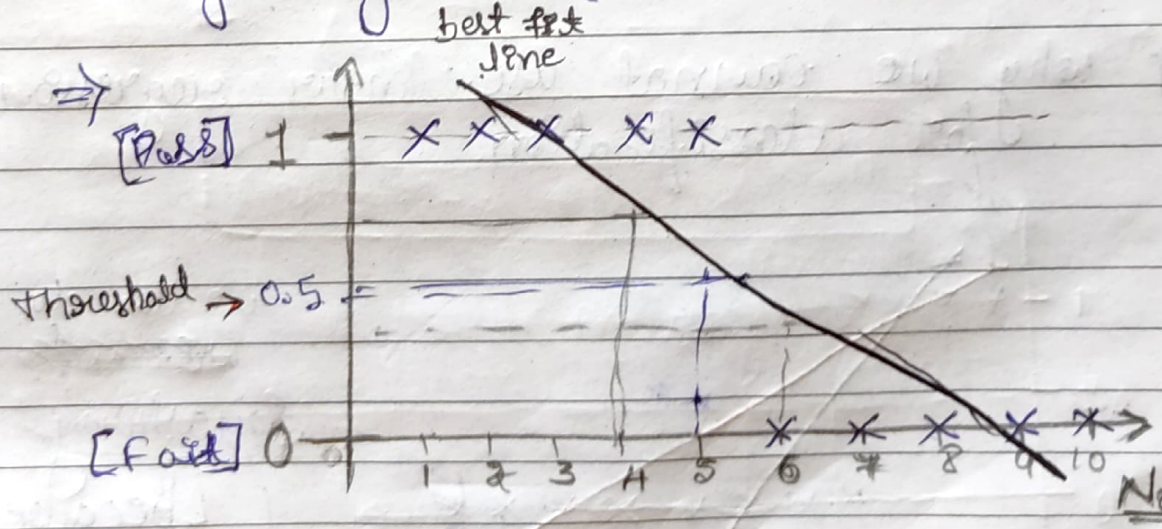


idea



Q) Can we solve this classification problem using regression?

⇒



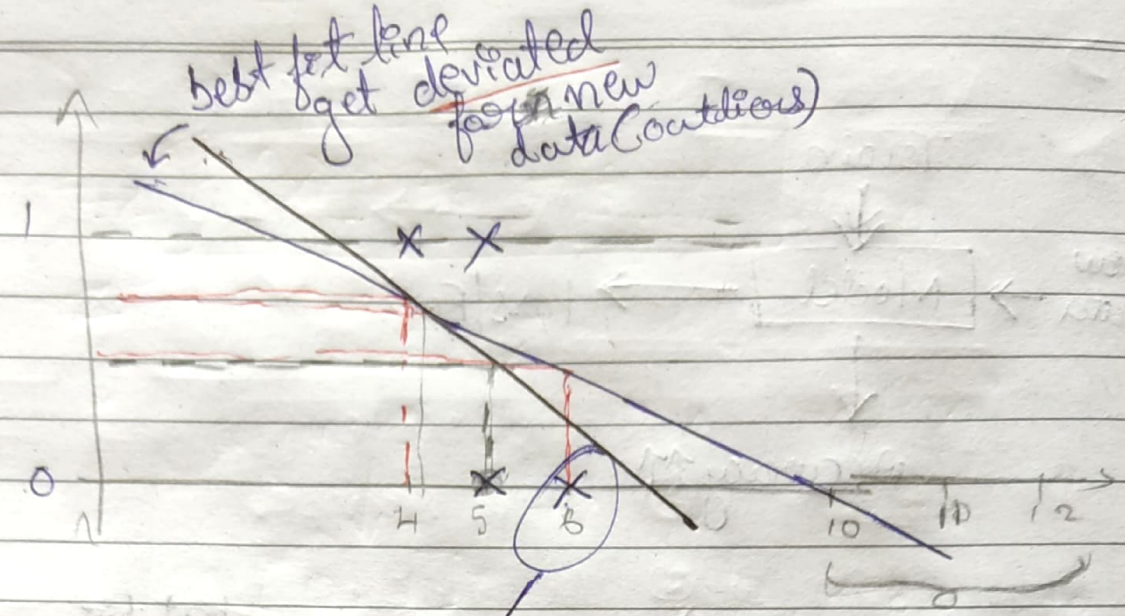
threshold

$\geq 0.5 \Rightarrow 1$

$< 0.5 \Rightarrow 0$

$x = 6 \Rightarrow < 0.5 = 0$

$x = 4 \Rightarrow > 0.5 = 1$



wrong prediction

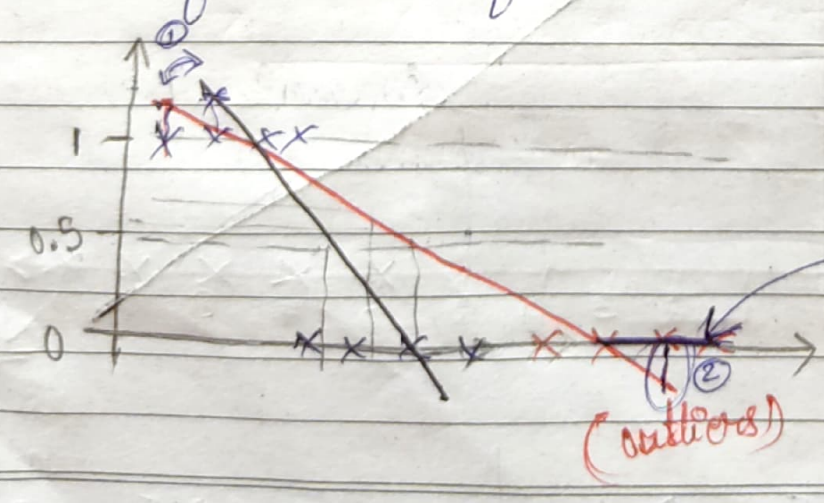
$6 \Rightarrow (\geq 0.5) = 1$

[pass]

[it's deviated]

So, we specifically don't use regression line.

Q) Why we cannot use linear regression for classification.



we can do "Squashing"

[because o/p is 0/1 but we are getting o/p >1 & <0. also.]

⇒

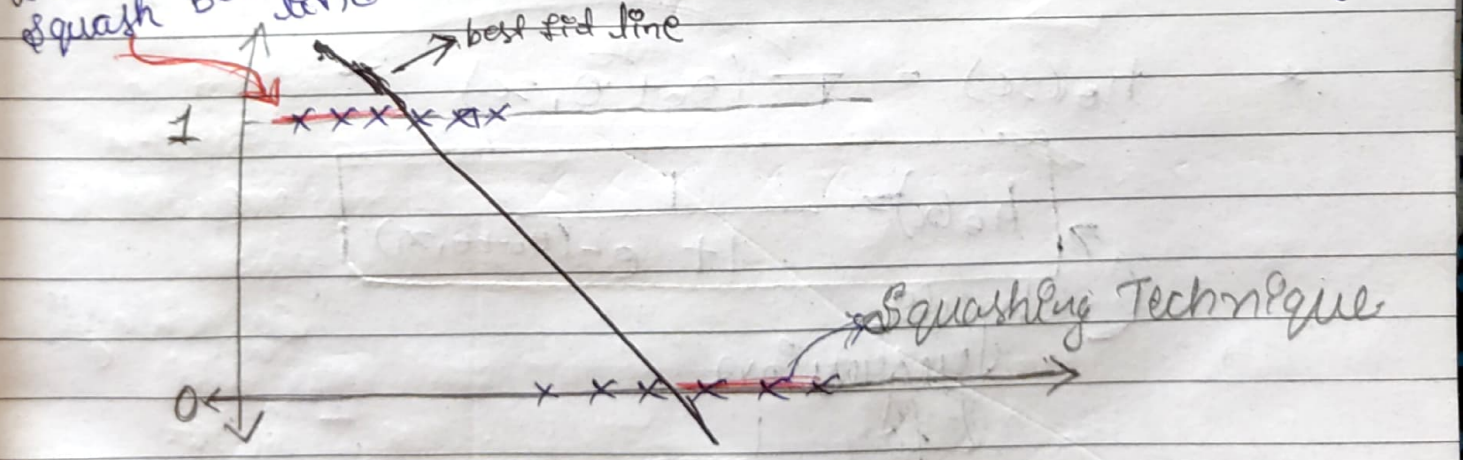
① Best fit line changes because of outliers
↳ Problem
Predict goes wrong.

② The outcomes comes (>1 and <0).
⇓
Solved be logistic regression
o/p
0 to 1
← has
→ done by squashing

Q) How logistic regression solves classification problem?

Aim
best fit line + squashing

⇒ we have to squash best fit line



best fit line → $h_\theta(x) = \theta_0 + \theta_1 x$

↓ squashing

"sigmoid activation function"

↳ This will transform the line to give output b/w 0 to 1

[0, 1]

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

where, $z = \theta_0 + \theta_1 x$

o/p : [0 to 1]

* $h_\theta(x) = \sigma(\theta_0 + \theta_1 x)$ → activation function responsible for "squashing" best fit line

$$h_\theta(x) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1)}}$$

Logistic Regression [for 1 independent & 1 dependent feature]

* Linear regression Cost function

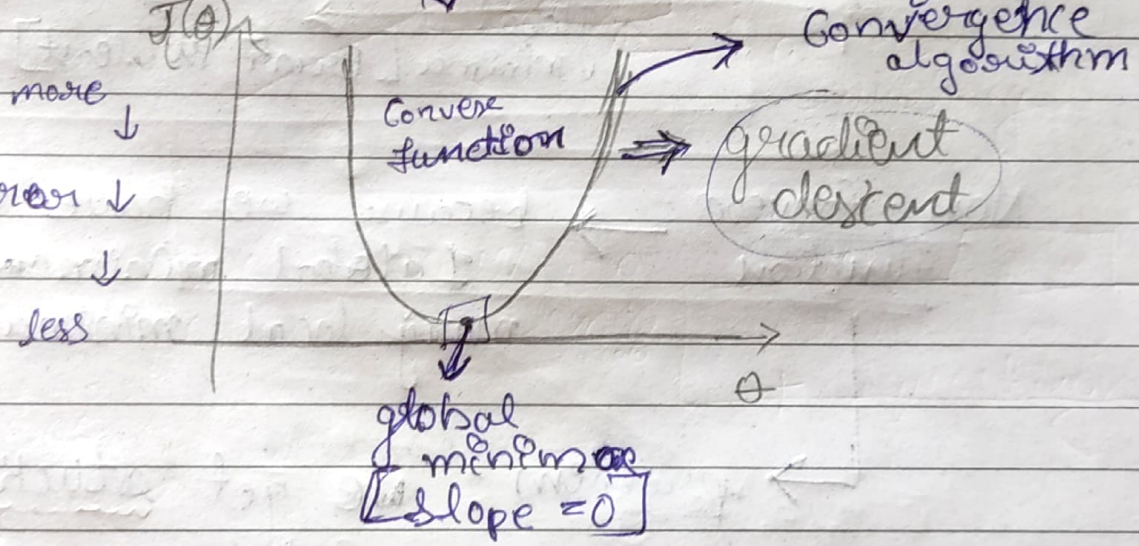
Cost function, $J(\theta_0, \theta_1) = \frac{\sum_{i=0}^{n} [y_i - h_\theta(x)]^2}{n}$ (actual − predicted)

↓ mean square error

$h_\theta(x) = \theta_0 + \theta_1 x_1$

* Logistic regression Cost function

used for Convergence algorithm



* Aim is to reach global minima.

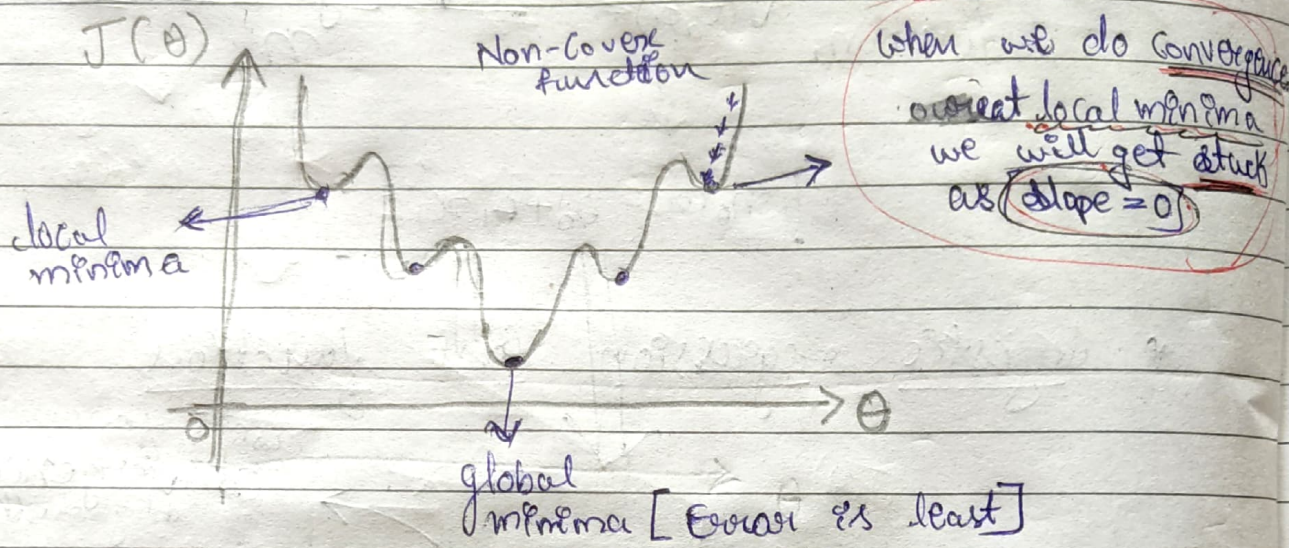
* logistic regression Cost function

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

$h_\theta(x) = \frac{1}{1 + e^{-[\theta_0 + \theta_1 x]}}$

↓ output 0 to 1

* we (donot use) this cost function of logistic regression



Non-Convex Function → because we have 1 global minima and many local minima

↳ problem: we get "stuck during converging" as "slope become 0 at local minima".

* So, Cost function which we use is called "log loss function".

This give only 1 global minima and no local minima

* log loss function

actual = y_i, predicted = $\hat{y}$ / $h_\theta(x)$

~~$J(\theta_0, \theta_1) = -(y_i) \log[h_\theta(x)_i] - (1)$~~

Cost Function

$$J(\theta_0, \theta_1) = -(y_i) \log[h_\theta(x)_i] - (1-y_i) \log[1 - h_\theta(x)_i]$$

where,

y_i = actual data [0/1]

$h_\theta(x)_i$ = predicted data.

if $y_i = 1$, $J(\theta_0, \theta_1) = \begin{cases} -\log[h_\theta(x)_i], & \text{if } y = 1 \\ -\log[1 - h_\theta(x)_i], & \text{if } y = 1 \end{cases}$

Final aim: Minimize cost function $J(\theta_0, \theta_1)$ by changing θ_0 & θ_1

* Convergence algorithm

Repeat unit Convergence

{

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta} J(\theta_0, \theta_1)$$

}

* α ⇒ learning rate

* Logistic regression with Regularization parameter

feature selection → L1
reduce overfitting → L2

↳ To include L1 and L2 regularization.

[like ridge (L2) and lasso (L1) with linear regression]

* Cost function, $J(\theta_0, \theta_1) = -y \log[h_\theta(x)] - (1-y) \log[1 - h_\theta(x)]$

$$h_\theta(x) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1)}}$$

$$J(\theta_0, \theta_1) = \begin{cases} -\log[h_\theta(x)], & \text{if } y = 1 \\ -\log[1 - h_\theta(x)], & \text{if } y = 0 \end{cases}$$

Regularization

① Reduce overfitting - [L2 regularization (ridge)]

$$J(\theta_0, \theta_1) = -y \log[h_\theta(x)] - (1-y)[1 - h_\theta(x)] + \text{L2 regularization}$$

$$\lambda \sum_{i=1}^{n} (\text{slope})^2$$

② Feature selection (lasso) — [l1 regularization]

$J(\theta_0, \theta_1) =$ ~~J~~ cost function + l1 regularization

$\lambda \sum_{i=1}^{n} |\text{slope}|$

③ Combination (elastic net)

$J(\theta_0, \theta_1) =$ cost function + l2 reg + l1 reg

$J(\theta_0, \theta_1) = -y_i \log[h_\theta(x)] - (1-y_i)\log[1-h_\theta(x)] + \lambda \sum_{i=1}^{n} (\text{slope})^2 + \lambda \sum_{i=1}^{n} |\text{slope}|$

> from sklearn.linear_model import LogisticRegression

we have parameter

Parameter

regularization ⇒ penalty = 'l1' / 'l2' / 'elasticnet' / None

$\lambda \Rightarrow C = 1.0$

how this 'C' related to "λ"?

$C \propto \frac{1}{\lambda}$

why specifically?

⇒ other algo we use 'C' so here also C

> regressor.fit(x_train, y_train)

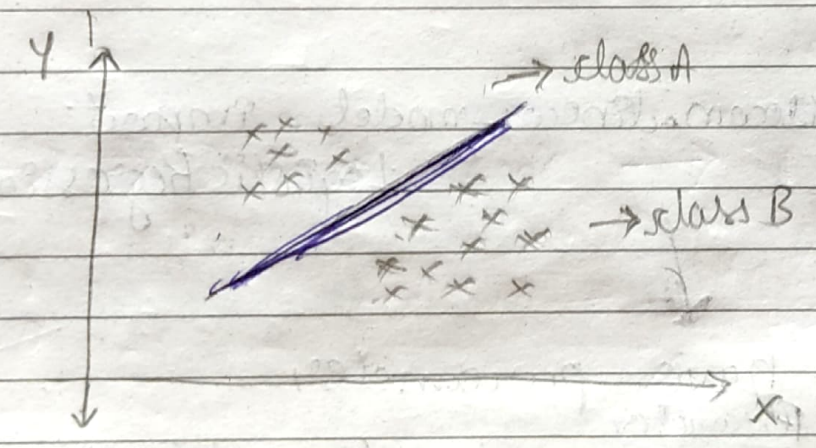
> regressor.predict(x_test)

> regress.score(x, y) → mean accuracy

> regress.predict_proba(x) → probability estimates

* Performance Metrics

① Confusion matrix. ② Accuracy. ③ Precision
④ Recall ⑤ F-Beta Score.

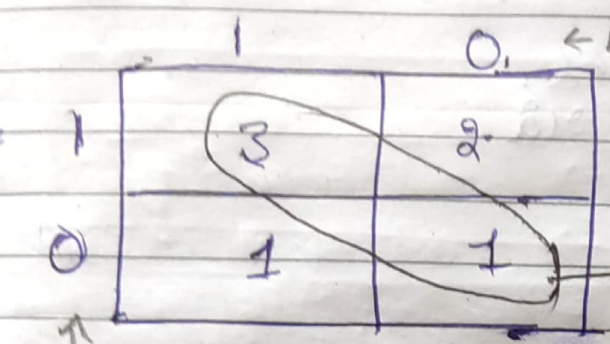


* Consider, Dataset ← actual value of dependent feature

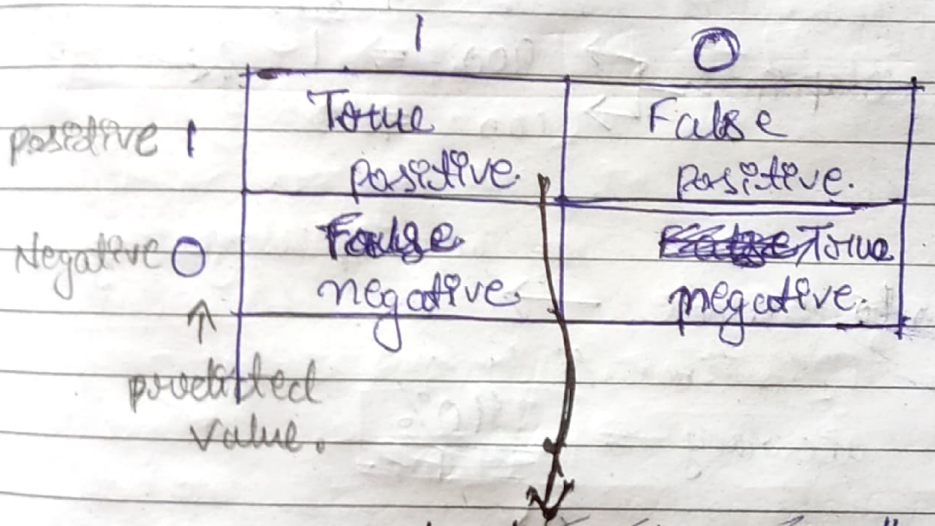
	x_1	x_2	y	ŷ
[W] Wrong prediction →	—	—	0	1
[C] Correct prediction →	—	—	1	1
C	—	—	0	0
C	—	—	1	1
C	—	—	1	1
W	—	—	0	1
W	—	—	1	0

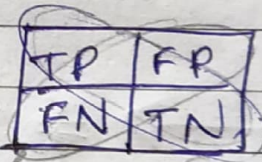
ŷ ← prediction

① Confusion matrix :



binary classification → Matrix (2×2)





actual value is "True" and prediction gives "positive" result. tells value is ↑

Model accuracy	=	TP + TN / TP + FP + FN + TN

$$= \frac{3+1}{3+2+1+1} = \frac{4}{7} = 57.1\%$$

② Accuracy

- R^2
- adjusted R^2

③ Precision

* Consider, —

Dataset → Imbalanced dataset

1000 datapoints → 900 ⇒ 1
→ 100 ⇒ 0

* Dumb model → 1 (what i/p but o/p always)
↳ biased towards 1 class

	1	0
1	900	100
0	0	0

accuracy = 90%

Even we get good results

So accuracy is not only thing to tell model is performing good

We use when FP [False positive] is important

where we have to FP ↓↓

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

out of all the actual values how many are correctly predicted.

	1	0
1	TP	FP
0	FN	TN

② Recall → when FN is important

FN ↓↓

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

Out of all the predicted values how many are correctly predicted actual value with

	1	0
1	TP	FP
0	FN	TN

usecase :

① Spam classification

Text → [Model] → Spam / Not Spam

actual \ predicted	1	0
1	TP	FP
0	FN	TN

TP ⇐ Scenario 1: Mail → Spam ✓ accurate
model predict → Spam ✓ Scenario

TN ⇐ mail → not Spam ✓ accurate
model → not Spam ✓ scenario
* if important mail is in Spam ✓ if u miss it it is a (blunder)

FP ⇐ (imp) mail → not a Spam ✓ Wrong
model → Spam scenario

FN ⇐ mail → Spam ✓ worst
model → not a Spam ✓ scenario
(not a blunder)
we can ignore Spam message

→ So we use "precision" (To calculate accuracy)

$$\frac{TP}{TP + FP}$$

---

* Usecase 2:

To predict whether a person has diabetes or not

Diabetes	Diabetes: 1	Diabetes: 0 (No Diabetes)
1	TP	FP
0	FN	TN

TP ⇐ person → diabetes } Correct
model → diabetes } scenario

TN ⇐ person → No diabetes } Correct
model → No diabetes } scenario

FP ⇐ person → No diabetes } wrong
model → Diabetes } scenario
(not a blunder)
as person taken care more

[imp] FN ⇐ person → Diabetes } wrong
model → No diabetes } scenario
(blunder)
because person health get

→ So, we use "recall". [To calculate accuracy]

$$\frac{TP}{TP+FN}$$

---

problem: Tommorrow the stock market will crash or not

we invest and loose

⇒ we have to Calculate "Recall" as "FN" is important because if our model says market "crash" (1) but our model predict market will "not crash" (0) ⇒ blunder

predict \ actual	crash (1)	not crash (0)
1	TP	FP
0	FN	TN

⑤ F-Beta Score

$$\text{F-Beta Score} = (1+\beta^2)\frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

Scenario

① If FP and FN are equally important

$\beta = 1$

$$\text{F1 Score} = (1+1)\frac{P*R}{P+R} = 2\frac{P*R}{P+R}$$

⇓

"Harmonic mean"

② If FP is more important than FN

$\beta = 0.5$

$$F_{0.5}\text{ Score} = (1+0.5)\frac{P*R}{P+R}$$

③ If FN is more important than FP

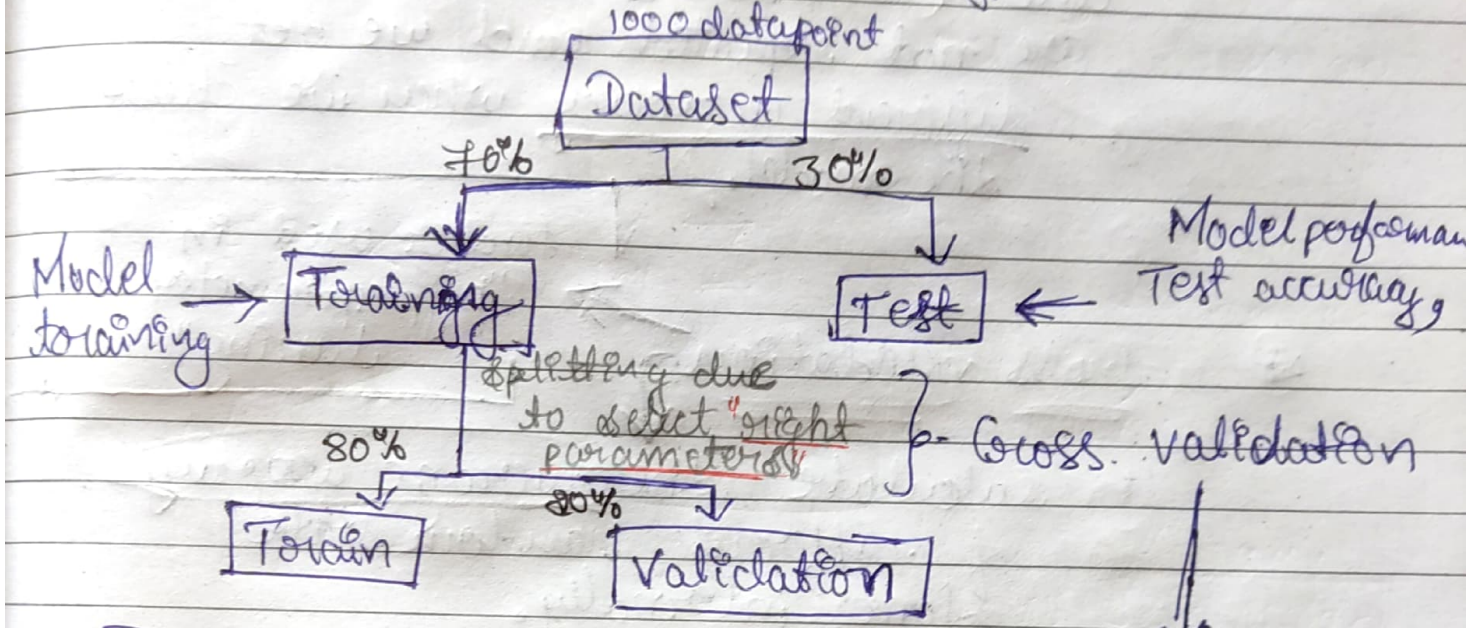
$\beta = 2$

$$F_2\text{ Score} = (1+4)\frac{P*R}{P+R}$$

* Cross Validation

→ Used to do hyperparameter tuning.

2 types



done to do "hyperparameter tuning"

* If we do split using train_test_split

we have a parameter "random_state"

↳ if we change value of this we get different values.

→ if =100, acc → 80%
=42, acc → 75%
= —, acc → 90%

} changes based on random_state value

So, instead of changing values again and again we use cross validation

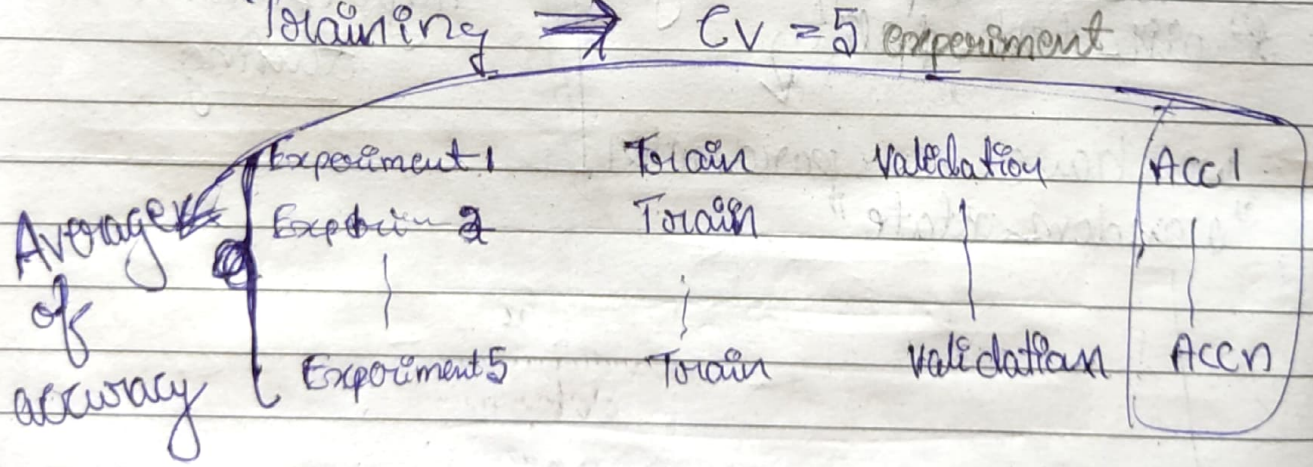
Instead of change random_state we use

In Cross validation it will internally divide training data into train and validation data and (it make sure the record are not repeated and we get different accuracy when we check the model.)

* Cross validation → further used in "hyperparameter tuning"

Calculate the average accuracy of different random state done internally.

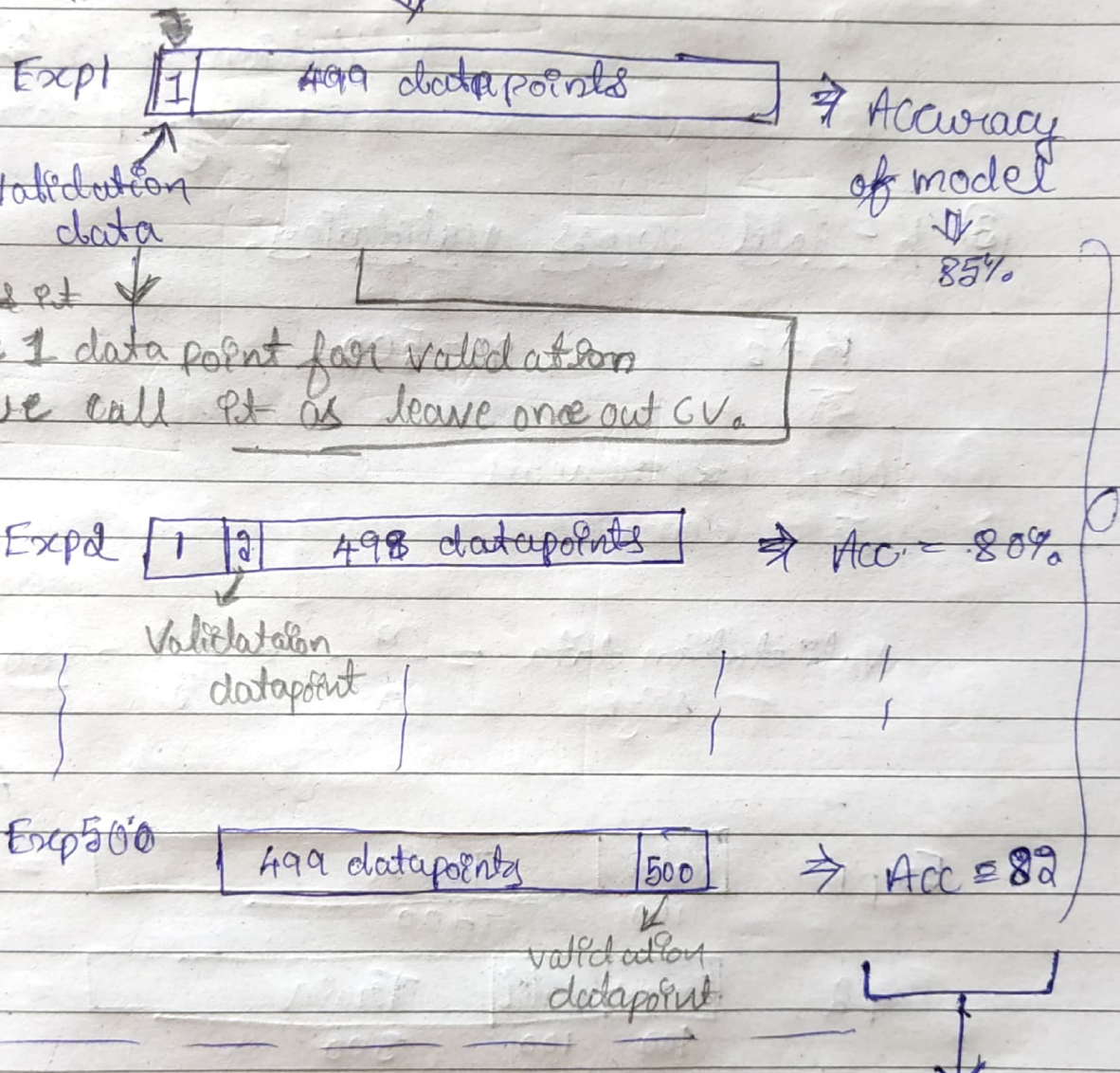
*



* Types of cross Validation

① leave one out cross validation [LOOCV]

Training Data → 500 datapoints



Now, we Calculate average of all the accuracy.

* Dis advantage:

① Time Complexity is huge for training large dataset.

② Model tends to overfit ⇒ Training accuracy ↑↑
New data → accuracy ↓↓

② leave p out CV :

p value can be anything based on problem

hyper parameter

p = 10, 20, ...

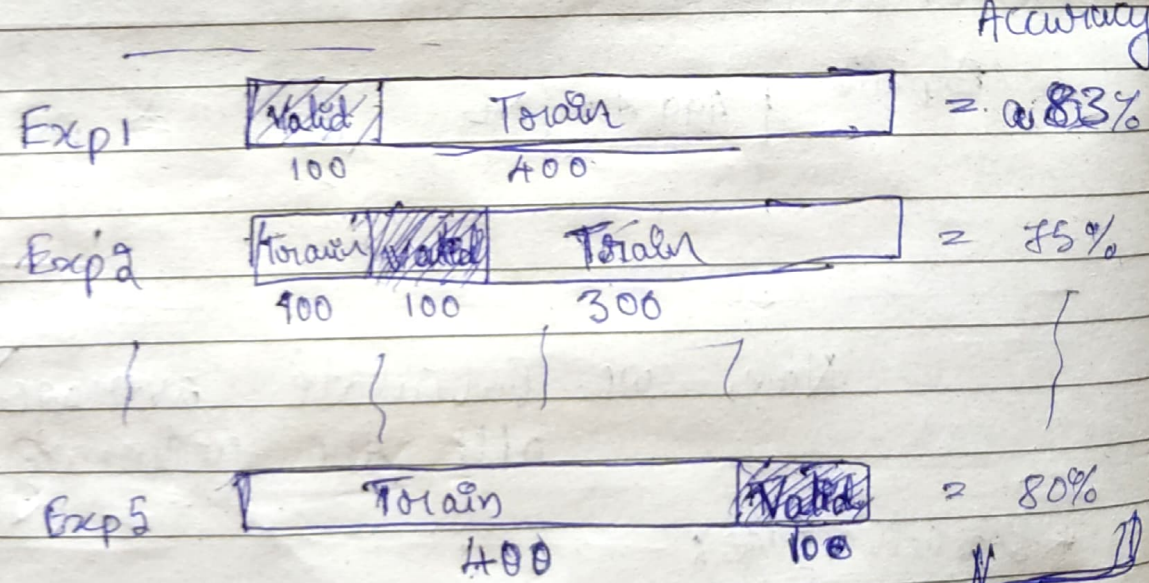
Validation points will be 'p' data points

③ K-fold Cross validation:

K = 5 , n = 500

No of experiments to be done

Validation data size $= \frac{n}{K} = \frac{500}{5} = 100$



Contains equal proportion of validation datapoints.

most used

Stratified K Fold CV:

Used for "Imbalanced dataset"

5 experiment

K = 5

n = 500

{ 1 → 350
0 → 150 }

Exp1 | Valid 100 | Train 400 | = Acc 1

Validation data contain Equal [0's = 1's] no of datapoints from class

{ 50 → 1's
50 → 0's }

100

Exp5 | Train | Valid | = Acc 5

Average of accuracy

* Most of the time we perform K fold validation.

* We use cross validation inside internally in the hyperparameter tuning to select the best parameters

* Hyperparameter tuning with cross validation

↳ finding the "best parameter" while training the models.

Types

1. GridSearch CV.
2. RandomizedSearch CV.

---

* Few parameters of LogisticRegression (Python)

- > penalty {'l1', 'l2', 'elasticnet', None}
- > solver {'lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', 'saga'}

Now, Task is to know which parameter to use and which combination

↓

So, we use hyperparameter tuning

---

1. Grid Search CV [GridSearch + Cross validation]

using ⇓

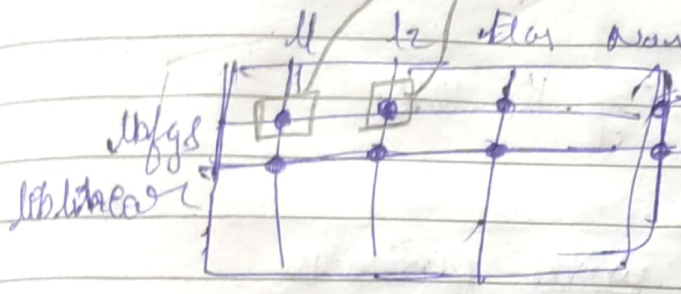
we try to find out best combination of parameter

⇓

by taking all possible combination

apply logistic regression

accuracy = 80% 85%



then we apply "Cross validation" internally and Calculate average accuracy.

*Here, using grid search CV we are going to find out all possible Combination with their accuracy and select the best Combination

② ~~Randomized Search CV~~

* Dis advantage:

① Time Complexity increases with large dataset. for Toning the model.

③ Randomized Search CV

Stratified K Fold CV + [CV=5]

[n_iter = 100]

* Here, we add iteration parameter which tell randomly select perform 100 iteration

10 different Combination

+

Cross validation

ex: Here, 10 different Combination + CV=5 ⇒ 50

select the best parameter

Advantage:

① Time Complexity is less

* Logistic Regression with python

> from sklearn.dataset import load_iris

> dataset = load_iris()

↓

dictionary contains information about dataset

> df = pd.DataFrame(dataset['data'], columns = dataset['feature_names'])

① Independent and dependent feature
(df.iloc[:, :-1])

② Train Test split (model_selection)

③ ~~Standard Scaler~~

④ Train model

> from sklearn.linear_model import LogisticRegression

> classifier.fit(x_train, y_train)

> y_pred = classifier.predict(x_test)

⑤ Predict from

⑥ Accuracy

> from sklearn.metrics import confusion_matrix, accuracy_score,

> accuracy_score (y_test, y_pred)

> classification_report (y_test, y_pred) ⇒ for F1 score, recall, precision

⑥ > from sklearn.model_selection import KFold

> cv = K.Fold (n_split = 5)

~~Note~~, > import cross_val_score

> cv_scores = cross_val_score (classification, xtrain, ytrain, scoring = 'accuracy', cv = cv)

classification → logistic regression model

xtrain, ytrain → telling cross validation to split data within given data

cv → output 5 different accuracy

> final_score = np.mean (cv_scores)

* Hyper parameter tuning with python

> from sklearn.model_selection import GridSearchCV

> classifier = LogisticRegression()

> clf = GridSearchCV(classifier, param_grid = parameters, CV = 5)

parameters = { 'C' : [...], 'penalty' : [...] }

internally we use KFold cross validation

> clf.fit(x_train, y_train)

split data into train & validation then training

> clf.best_params_

> Calculate accuracy, precision, recall

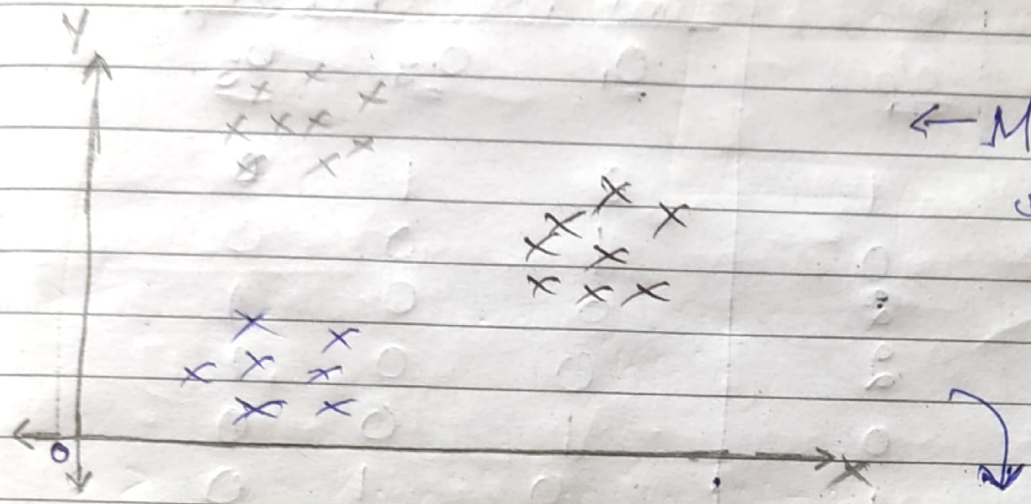
model_selection

> Import : RandomizedSearchCV

> clf = RandomizedSearchCV(LogisticRegression(), param_distributions = parameters, CV = 5, n_iter = 20)

estimator

* Logistic Regression for ~~multiclass~~ Multiclass classification



← Multiple class

2 Technique to classify

① OVR → One versus rest

② Multinomial. →

← we cannot classify using a single best fit line

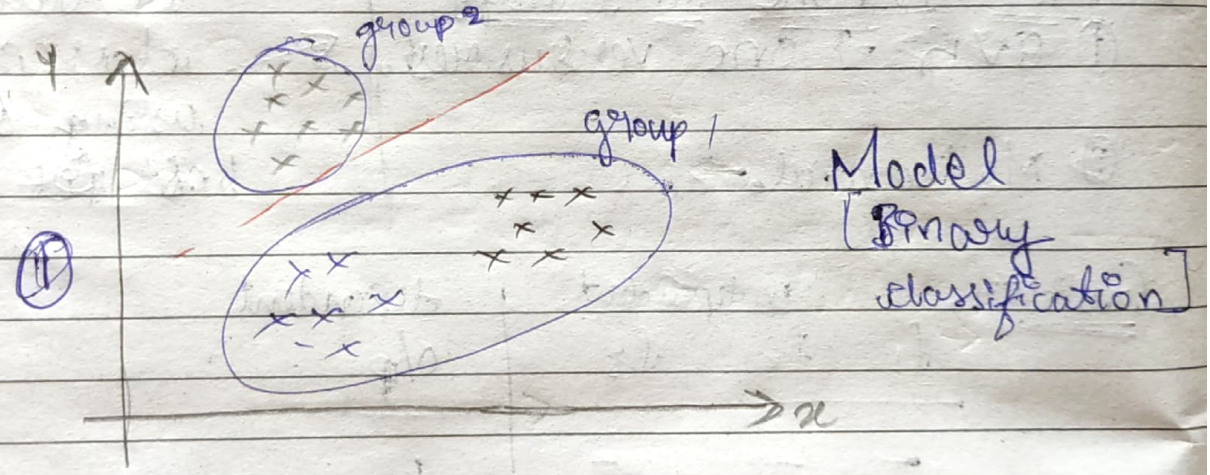
Consider,

independent			dependent
f_1	f_2	f_3	o/p
—	—	—	1
—	—	—	0
—	—	—	2
—	—	—	2
—	—	—	0
—	—	—	1

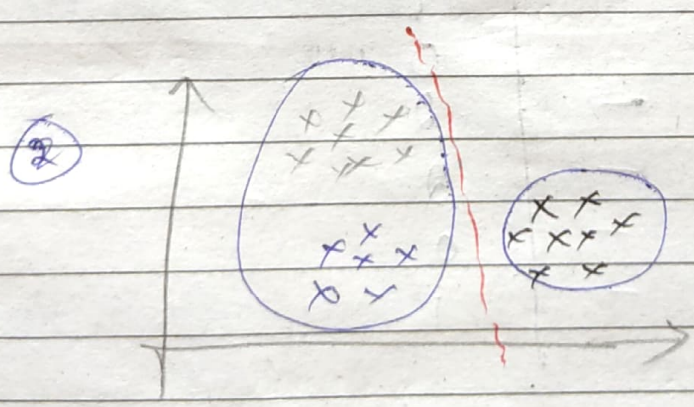
↳ 3 categories

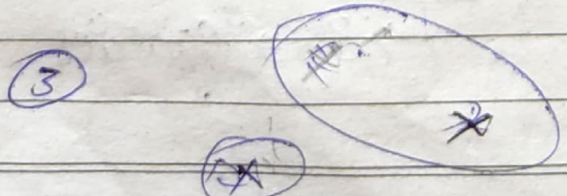
(1) OVR [one versus rest]:

o/p	o/p	One hot encoding O_1	O_2	O_3
	1	0	1	0
	0	1	0	0
	2	0	0	1
	2	0	0	1
	0	1	0	0
	1	0	1	0



Model [Binary classification]





M1 model → I/p = f1 f2 f3 ; O/p = θ1

⇓

Binary classification

M2 model → I/p = f1 f2 f3 ; O/p = θ2

M3 model → I/p = f1 f2 f3 ; O/p = θ3

* Works similarly if we have many classes

* New Test data → [M1 M2 M3]

O/p : [0.25, 0.25, 0.5] Σ=1

⇓

highest probability

O/p : θ3 → [2]

> In Logistic regression we have parameter

↓

> multi_class : {'auto', 'ovr', 'multinomial'}

default

'ovr' → we get probability output

'multinomial' → we get probability

M1 model → o/p = f1 f2 f3 ; o/p = 01

↓ Binary classification

M2 model → o/p = f1 f2 f3 ; o/p = 02

M3 model → o/p = f1 f2 f3 ; o/p = 03

* Works similarly if we have many class

* New Test data → [M1 M2 M3]

o/p : [0.25, 0.25, 0.5] Σ=1

↓ highest probability

o/p : 03 → [2]

> In logistic regression we have parameter

↓ default

> multi_class : {'auto', 'ovr', 'multinomial'}

'ovr' → we get specific output.

'multinomial' → we get probability

* Usually, in logistic regression we get less accuracy as points will be scattered.

* python it's same

① x, y [segregate dependent & independent]

② train test split

Hyper parameter Tuning ⇒

③ Model training

> classifier = LogisticRegression(
multi_class = 'ovr',
solver = 'lbfgs')
↳ optimizer

④ prediction

> model.predict_proba(x_test)

⑤ accuracy, confusion matrix

> from sklearn.datasets import make_classification

> x, y = make_classification(n_samples = 1000,
n_features = 10, n_informative = 5,
n_redundant = 5, n_class = 3,
random_state = 1)

* WSGI → Web Server Gateway Interface

* End to End project :

Folders

- ① .ebextensions
  - python.config
- ② dataset
  - diabetes.csv
- ③ notebooks
  - jupyternotebook.ipynb
- ④ model
  - pickle file
    - eg: model.pkl
- ⑤ templates
  - index.html, home.html, ...
- ⑥ static
  - css
    - main.css, style.css, ...
- ⑦ * requirements.txt
- ⑧ * application.py
- ⑨ * data.json

go back one step

* '../dataset/data.csv' ⇐ path

* In a html ↓ sending info to flask

< form action = "{{ url_for ('predict_datapoint') }}" method = 'POST' >

(or)

< form action = "/predict" method = 'POST' >

function

route

* Flask ↓

if request.method == 'POST':

result = int(request.form.get('---'))

* sending info to html.

Flask → return render_template ('predict.html', result = result)