Naive Bayes and Ensemble Technique HandWritten Notes

- Darshan R M

"Learn, Share, and Collaborate"

Week 17

Naive Bayes and Ensemble Technique and types

* Naive Bayes algorithm
  ↳ classification.

① Probability.
② Bayes Theorem.

* Events
  |
  ① Independent events
  ↳ Probability of happening of one event don't affect other.
  (or) happening

Ex: Rolling a dice ⇒ {1, 2, 3, 4, 5, 6}

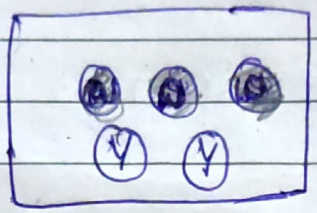
then

$$P(1) = \frac{1}{6}, \quad P(2) = \frac{1}{6} \text{ ---}$$

(ii) Dependent events

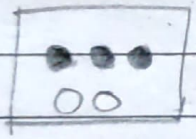
↳ Probability of happening an event affect the happening of other event.

Ex:



3 → orange
2 → yellow

① what is the probability of removing a orange marble and then a yellow marble.

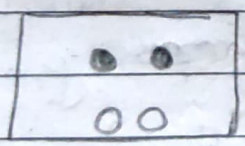


$n=5$

$P(O) = \frac{3}{5}$ ⇒ 1st event

Total orange marble / Total marble

orange has been removed



$n=4$

$P(Y) = \frac{2}{4}$ ⇒ 2nd event

$P(Y/O)$

Probability of orange marble then yellow marble

1st event, 2nd event

$P(O \text{ and } y) = P(O) * P(Y/O)$

(conditional Probability) $P(Y/O)$ = Probability of yellow marble when orange marble has been taken out.

$P(\text{0 then } y) = P(0) * P(Y/0)$

$= 3/5 \times 1/2 = \frac{3}{10}$

Generally

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

$P(A \text{ and } B)$ → Probability of happening of event A and then happening of event B.

$P(B/A)$ → Probability of event B when event A has completed/happened.

* Baye's theorem

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

$P(A) * P(B/A) = P(B) * P(A/B)$

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

$P(A/B)$ → Probability of event A when event B has already happened.

$P(A)$ → Probability of event A

* Dataset

x_1	x_2	x_3	Y
–	–	–	Yes
–	–	–	No
–	–	–	Yes
–	–	–	No

(independent feature; dependent feature)

Now, we solve this entire prediction by using Bayes theorem.

⇒ The algorithm which use bayes theorem is called "Naive bayes algorithm".

$$\Rightarrow P\left(\frac{y}{(x_1, x_2, x_3)}\right) = \frac{P(y) * P(x_1, x_2, x_3)/y}{P(x_1, x_2, x_3)}$$

Probability of y given that x_1, x_2 and x_3

$$P\left(\frac{y}{x_1, x_2, x_3}\right) = \frac{P(y) * p(x_1/y) * p(x_2/y) * p(x_3/y)}{p(x_1) * p(x_2) * p(x_3)}$$

For "Yes",

$$P\left[\frac{Yes}{(x_1, x_2, x_3)}\right] = \frac{P(Yes) * P\left(\frac{x_1}{yes}\right) * P\left(\frac{x_2}{yes}\right) * P\left(\frac{x_3}{yes}\right)}{P(x_1) * P(x_2) * P(x_3)}$$

Constant

If [0.60, 0.40] then conclusion/output is "yes"

(Y, N)

For "No",

$$P\left[\frac{No}{(x_1, x_2, x_3)}\right] = \frac{P(No) * P\left(\frac{x_1}{No}\right) * P\left(\frac{x_2}{No}\right) * P\left(\frac{x_3}{No}\right)}{P(x_1) * P(x_2) * P(x_3)}$$

Constant, So, we can remove this denominator.

Q) Consider dataset

Day	Outlook	Temperature	Humidity	Wind	Play Tennis
1					No
					No
					Yes
					Yes
14					

From 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
	Σ = 9	Σ = 5	Σ = 1	Σ = 1

Temperature

	Yes	No	P(E/yes)	P(E/No)
Hot	2	2	2/9	2/5
Mold	4	2	4/9	2/5
Cool	3	1	3/9	1/5
	Σ = 9	Σ = 5		

tell probability of Cool temperature when play tennis Yes No.

Play

		P(Yes)	P(No)
Yes	9	9/14	
No	5		5/14

∞ If: Y/N

Test (Sunny, Hot) → ?

O/P

$$\Rightarrow P\left(\frac{Yes}{(Sunny, hot)}\right) = \frac{P(Yes) * P\left(\frac{Sunny}{Yes}\right) * P\left(\frac{Hot}{Yes}\right)}{P(Sunny) * P(hot)}$$

Constant

$$= \frac{\left(\frac{9}{14}\right) \times \left(\frac{2}{9}\right) \times \left(\frac{3}{9}\right)}{\text{Constant}}$$

$$= \frac{2}{63} = 0.031$$

$$\Rightarrow P\left(\frac{No}{(Sunny, hot)}\right) = \frac{P(No) * P\left(\frac{Sunny}{No}\right) * P\left(\frac{Hot}{No}\right)}{\text{Constant}}$$

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

$$= \frac{3}{35} = 0.085$$

$$\Rightarrow P\left(\frac{Yes}{(Sunny, hot)}\right) = \frac{0.031}{(0.031 + 0.085)}$$

$$= 0.27$$

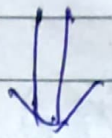
$$= 27\%$$

$$P\left(\frac{No}{(Sunny, hot)}\right) = \frac{0.085}{0.031 + 0.085}$$

$$= 0.73$$

$$= 73\%$$

outlook	Temperature	o/p
Sunny	Hot	73% ⇒ play Not Tennis
		27% ⇒ They will play Tennis



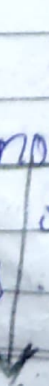
0 ⇒ person is not playing Tennis

* Variants of Naive Bayes

1. Bernoulli Naive Bayes.
2. Multinomial Naive Bayes.
3. Gaussian Naive Bayes.

1. Bernoulli Naive Bayes :

⇒ whenever your features are following a Bernoulli distribution, then we need to use Bernoulli naive bayes algorithm.



when there is only 2 outcome [0/1]

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

→

f_1	f_2	f_3
1	1	1
1	0	0
0	1	1
1	0	0

all features following Bernoullis distribution

Sparse Matrix

we use NLP technique

② Multinomial Naive Bayes:

⇒ i/p ⇒ in form of Text and

o/p ⇒ classification

Dataset : Spam classification.

Email	Spam / Not Spam
Baby	
You won 100 Lakh	Spam
You are good	Ham

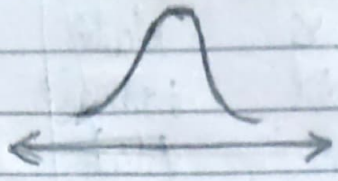
Convert into "numerical" values ⇒ using NLP [Natural language processing]

Inside we will have a formula which converts into numerical:

① Bow
② Tf-idf
③ Word2vec

(3) Gaussian Naive Bayes:

If the features are following Gaussian distribution, then we use gaussian naive bayes.



Iris dataset → features are Continuous

Age	Height	weight	overweight Yes / No
25	170	78	Yes
38	160	75	No
40	150	60	Yes
51	170	35	No

* We can use Bernoulli's and Multinomial naive bayes in NLP techniques

* Implementation

> from sklearn.datasets import load_iris

> from sklearn.model_selection import train_test_split

> x, y = load_iris(return_X_y = True)

① > train_test_split → 30%

② Model building

> from sklearn.naive_bayes import GaussianNB

> gnb = GaussianNB()

> gnb.fit(x_train, y_train)

> gnb.predict(x_test)

③ Performance matrices

* Azure deployment
  create
  Web App Service.

↓

Configuration required for github will be taken care by azure.

Can see
* github action ↓

Build and deploy section

* Ensemble Techniques and Bagging
  → Used to increase accuracy and efficiency of the model

① What is Ensemble ?

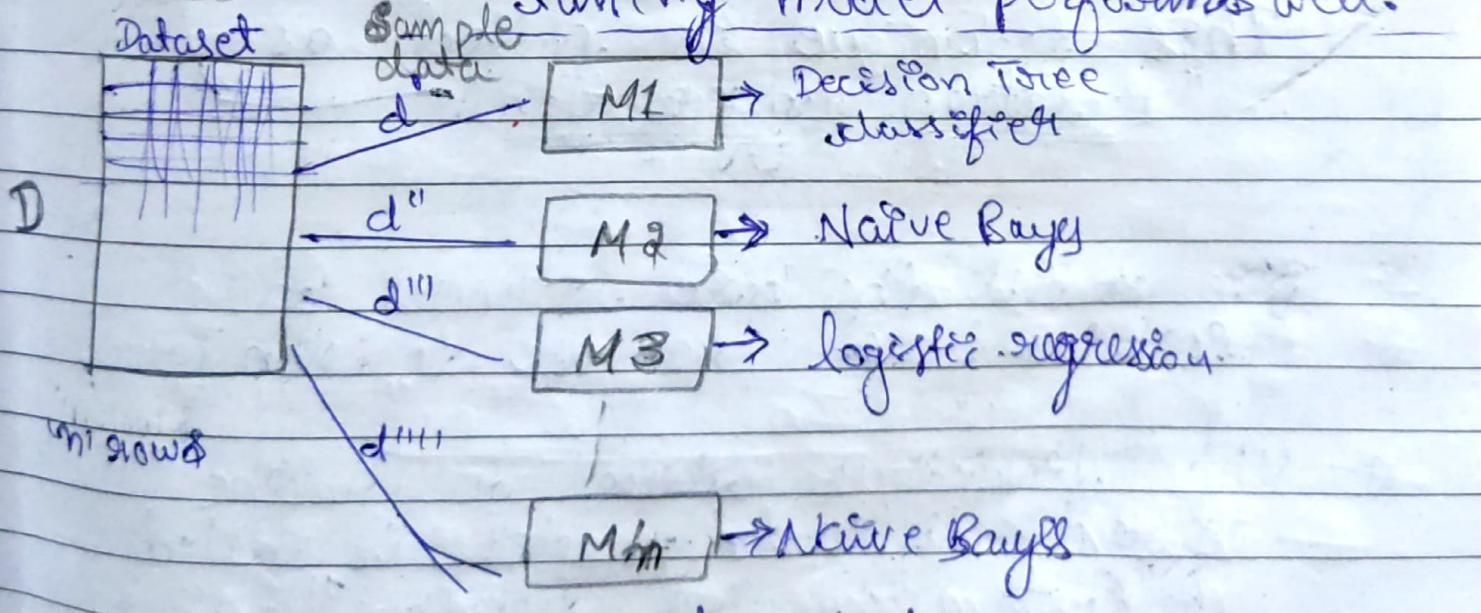
⇒ Combining multiple models → Train → Prediction

Two types

① Bagging.
Technique
(i) Random Forest classifier and regressor

② Boosting.
Technique
(i) Ada Boost.
(ii) Gradient Boost.
(iii) Xg Boost.

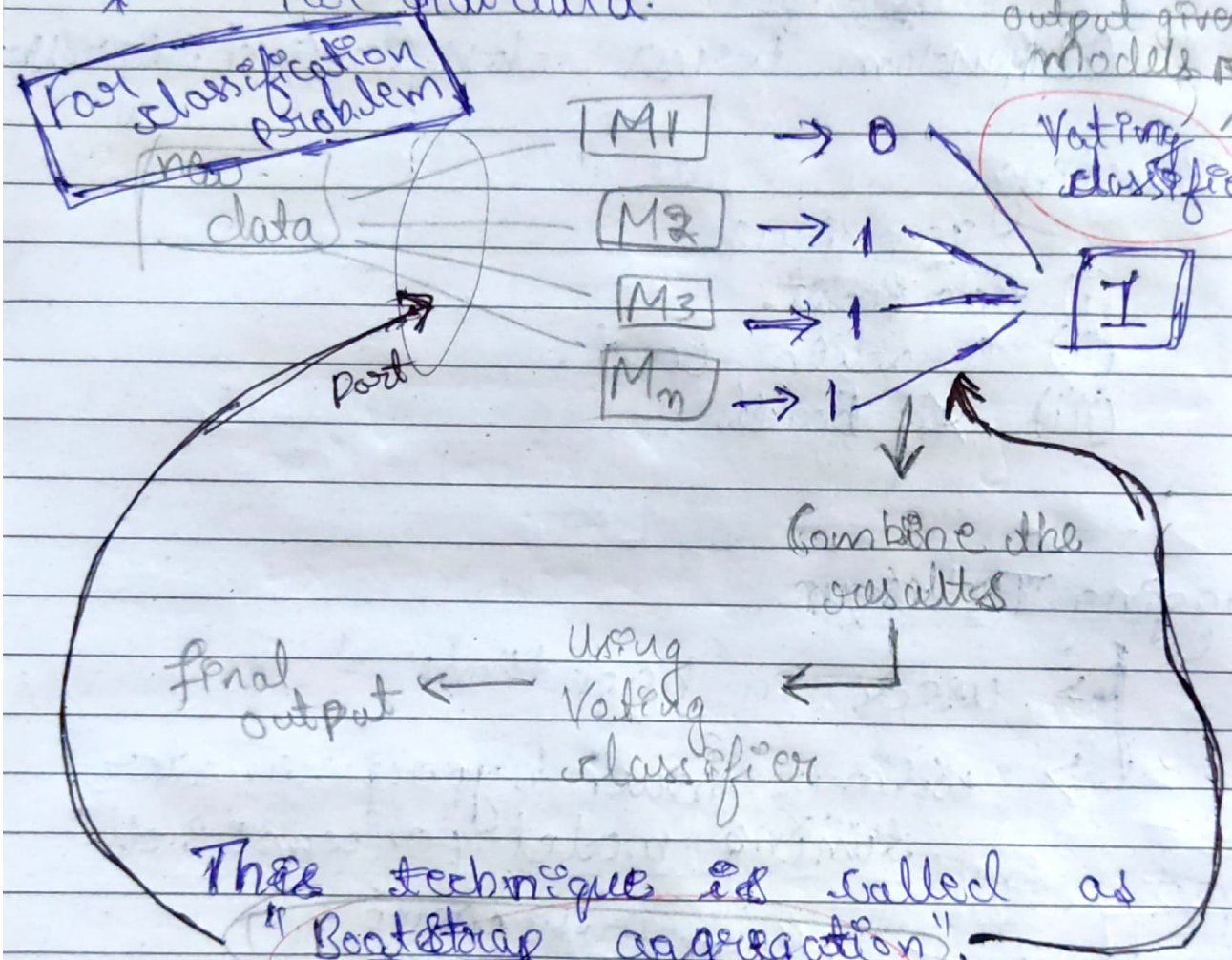
Bagging Technique
→ used in hackathons
→ without much hyperparameter tuning model performs well.



Bagis learners

* After training base learners with there corresponding sample data.

* For new data.



This technique is called as "Bootstrap aggregation".

* All the base learners of random forest we use decision Tree.

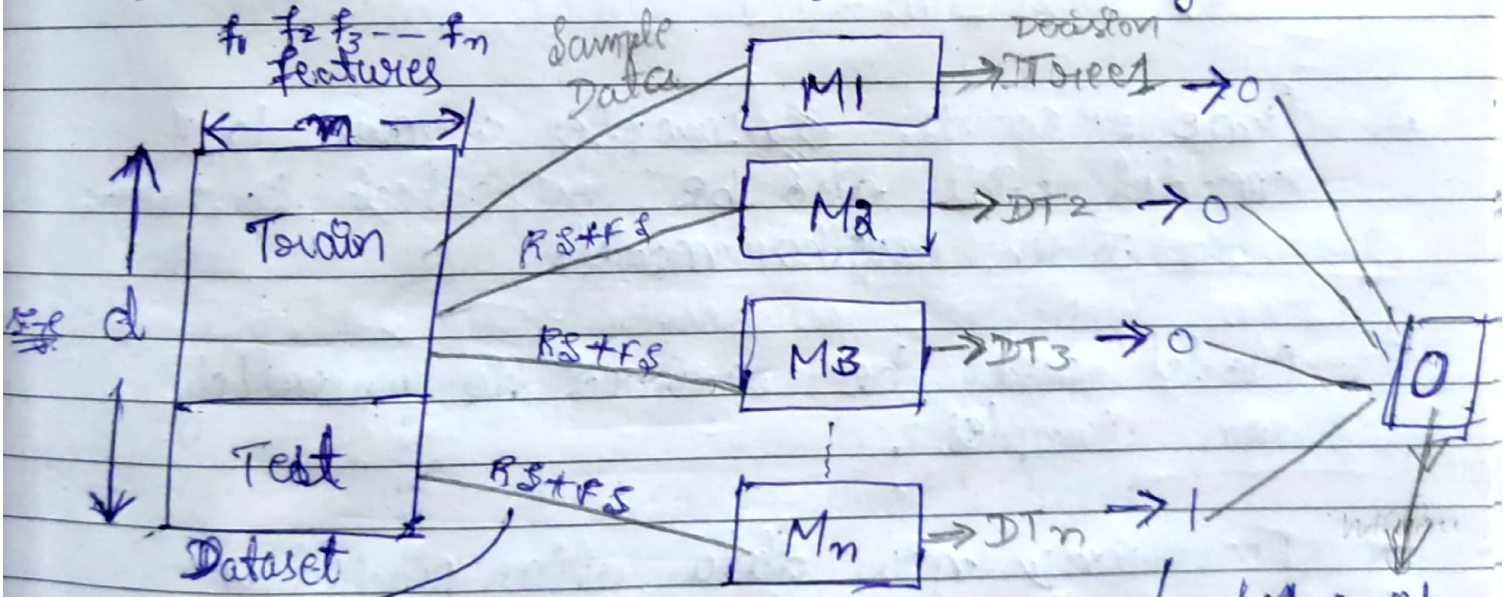
* The technique where we combine different different model is called as "Custom bagging technique"

For regression problem.

basis learner. → will be regression model

we get output : continous data then we will take average of all output to do the prediction.

* Random Forest classifier and regressor



Row sampling + Feature sampling

all this models are decision Tree

basis learners

Majority voting classifier

If its regression then we will take "average"

- Row-sampling ⇒ From the train data we take some records and give to model for training.

  ↓

  if 'd' samples

  ↳ then [d' < d]

- Feature sampling ⇒ From the train data we take particular feature for model training.

  ↓

  if 'm' samples

  ↳ then [m < n]

- Some records samples given to 1 model will also be repeated to taken to train another model.

- Model will be trained to parallely on samples.

- For any new data after prediction we will get output so we will take "majority voting classifier"

- This technique used in hackathon and Kaggle Competation. because gives good results always.

Notes:

classification → majority voting classifier.

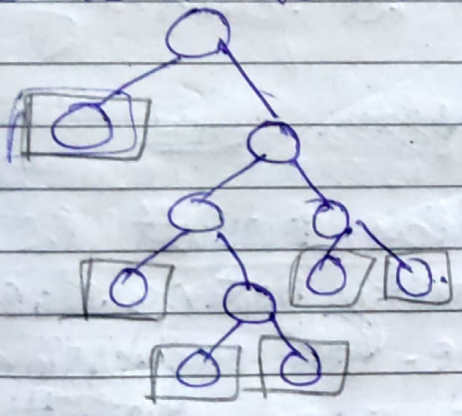
Regression → average o/p of all the models.

V. Imp

Q) why should we use random forest instead of decision Tree?

⇒ In DT we can also do post pruning and pre-pruning [using hyperparameter tuning]

Decision Tree



→ we take feature and split until we get leaf node.

↓

but if we split until leaf node it leads to "overfitting".

training → acc ↑ ↑ ← low bais and
test → acc ↓ ↓ high variance

but To get "generalized model".
we need to get "low bais and low~~high~~ variance".

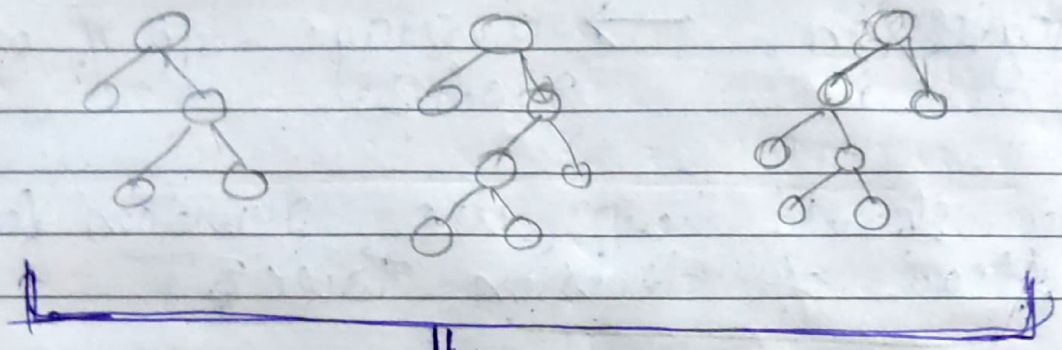
convert

Possible only using "random forest"

different decision Tree

* In Random forest Basic learners are trained parallely. → so we get parallel expertise

Generalized model



These are different decision Tree which are good at for thing

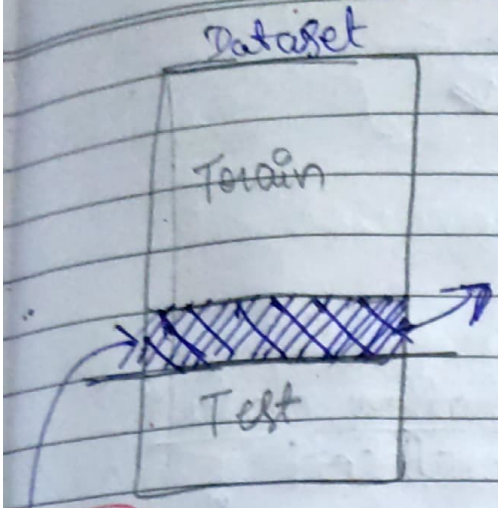
~~for whole~~ ~~prediction~~

So our prediction result is taken from majority voting classifier / average (in regression) we get good result

* Out of Bag Score Decision Tree [imp]

Interview asked

In random forest while training model we may get a ~~some~~ missing some records for training that data is called "Out of Bag Data". To use that data we have a parameter ~~oob score~~ when it is True Out of Bag data will be used as Validation data to calculate accuracy and performance of model



Some part of training data will not be selected for training

⇓

why ?

⇒ as we take random sample from data. there is a possibelety that some of record will not be selected for training.

Out of Bag data

that data is called

---

if this data is not select for training bcis learners. because of random sampling / feature sampling

we are losing this data.

in term we are hammering the performance of model by this data.

* So, while using random forest will get this scenario.

* There is 1 parameter ↘
out of bag score
oob_score" = True

↓

then out of Bag data will be considered as "validation data" and also used to check "accuracy & performance"

↓

these results are called as "oob-score"

can be : 85%, 90%, 75%

In python,

> from sklearn.ensemble import RandomForestRegressor / RandomForestClassifier

parameters ↙

① n_estimators → No of trees in forest
↳ 100 default

② criterion → ~~entropy, gini, log_loss~~

③ oob_score → (default → False)
↓
; out of Bag data ignored by default.

↓
many others

Production grade application understanding.

imp

* Implementation with pipelines and Hyperparameter tuning

Vimp: while building production grade application.

Until, we were doing Feature ~~detection~~, engineering and handling different value then training the model.

this model training is not one time activity because when we get "new data" we will retrain the model.

→ So, for that we cannot keeps on do feature engineering separately.

So, we need to write code when can automate the entire process

① Tips dataset.

② Label Encode → 'Time' feature

③ Segregate ~~Encode Test~~ & independent & dependent.

④ Train Test split.

missing values

> from sklearn.impute import SimpleImputer

> ~~from sklearn.preprocessing~~ → OneHotEncoder
Categorical → StandardScaler
Feature Scaling ← Numerical

(5) Pipeline

* we can automate process.
* we can combine multiple steps

> from sklearn.pipeline import Pipeline

> from sklearn.compose import ColumnTransformer

used to combine several feature extraction mechanism into single transformer.

(i) Indentify categorical and numerical columns.
> cat_cols = [_ , _ , _]
> num_cols = [_ , _ , _]

(ii) Feature engineering automation.

Pipeline

```
> num_pipeline = Pipeline(
      steps = [
          ('imputer', SimpleImputer(strategy='median')),
          ('scaler', StandardScaler())
      ]
  )
```

step to be done

name — object

> cat_pipeline = Pipeline('most_frequent'
steps = [
{'imputer', SimpleImputer(strategy = ____)},
{'onehotencoder', OneHotEncoder()},
]
)

* Now, as we use both pipline we have to wrapp them with 'ColumnTransformer'

wrapper
> preprocessor = ColumnTransformer(
[
('num_pipe', num_pipeline, numerical_cols),
('cat_pipe', catepipeline, categorical_cols)
]
)
name pipeline name features name

x_train = preprocessor.fit_transform(x_train)

⑥ Model Building + Evaluation

> from sklearn.ensemble import RandomForestClassifier.

* Automating model training process.

* models = {

'Random Forest': RandomForestClassifier(),

}

* function which ~~calculate~~ ~~the~~ do model training and evaluation

def evaluate_models (x_train, y_train, x_test, y_test, models):

report = { }

for i in range(len(models)):

Taking model. model = list(models.values())[i]

training model.fit(x_train, y_train)

predicting y_pred = model.predict(x_test)

Calculating accuracy model_score = accuracy_score(y_test, y_pred)

storing accuracy. report[list(models.keys())[i]] = model_score

return report

(7) Hyperparameter Tuning

> from sklearn.model_selection import RandomizedSearchCV

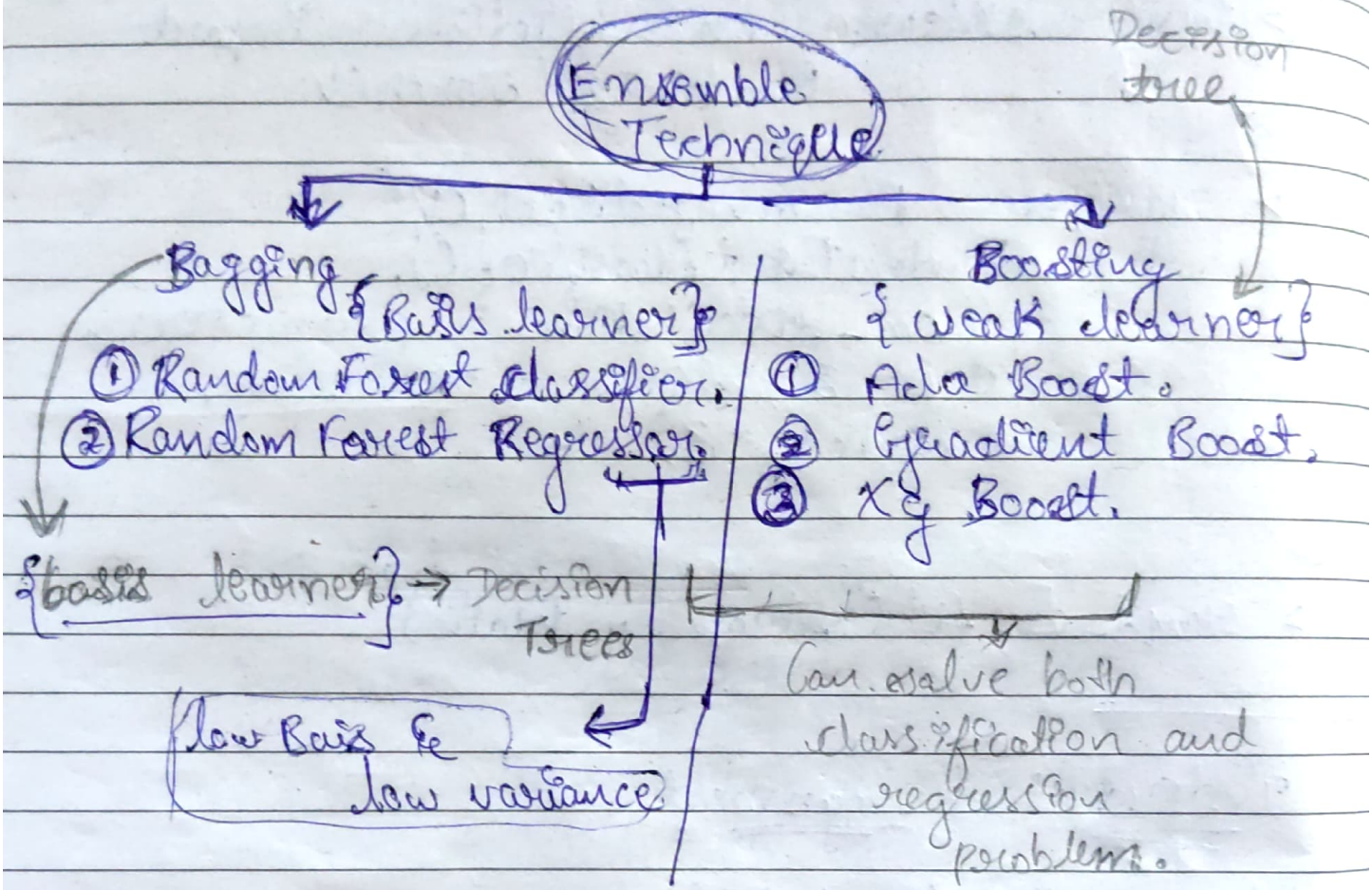
> rand_cv = RandomizedSearchCV(RandomForestClassifier(), param_distributions = parameters, cv = 5, scoring = 'accuracy', verbose = 3)

> rand_cv.fit(x_train, y_train)

Interval assignment

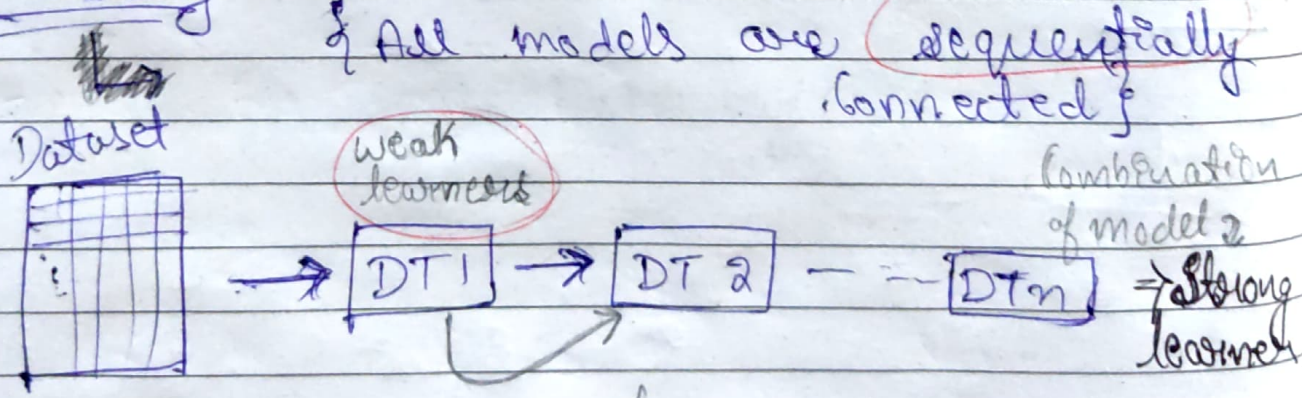
o/p : total bill.

* AdaBoost



* Boosting

{All models are sequentially connected}



* weak learner → haven't learnt much from the training dataset.

* Random Forest → classification problem → Majority voting classifier
→ regression problem → average of o/p of base learners

⇓

* Ada Boost → we Assign weights to the weak learners.

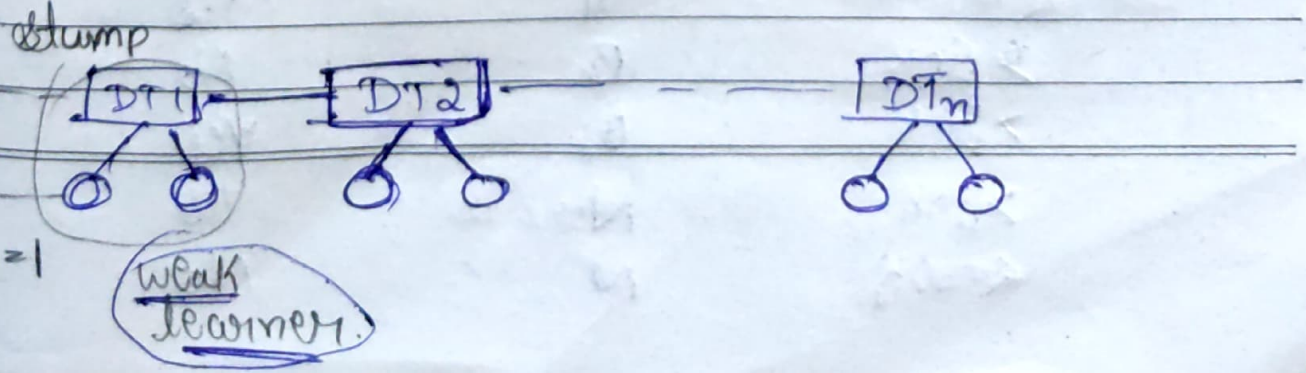
$$f = \alpha_1(M_1) + \alpha_2(M_2) + \cdots + \alpha_n(M_n)$$

both classification and regression

{ M_1, ... M_n } → Decision tree stumps

{ α_1 to α_n } → weights.

* Decision Tree stump →
A tree with 1 depth.



** weak learners [Decision tree stump]

↳ Underfitting ↓

40% Training → acc ↓↓ → High bais and
45% Testing → acc ↑↑ → High variance
(or) acc ↓↓

* when we combine sequentially multiple decision tree stump .we get a "strong learner"

↓

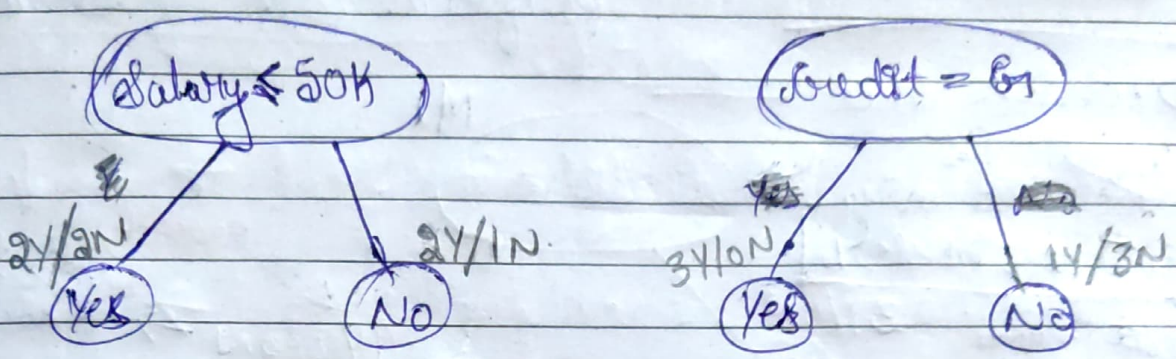
{ High Bais, High variance } → { low Bais, High variance }

Decision Tree stump

** Ada Boost maths

Salary	Credit Score	Approval
<=50K	Bad	No
<=50K	Good	Yes
<=50K	G	Yes
>50K	B	No
>50K	G	Yes
>50K	Normal	Yes
<=50K	N	No

Step 1: We create Decision Tree stump & select best one



DT stump-1 DT stump-2

* we can create other DT stumps also
* Now, From above 2 stumps which one we have to select ?

we use Impurity check

Entropy ↔ Gini

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

$$gini = 1 - \sum_{i=1}^{n} P_i$$

* In DT stump-1 → Yes ⇒ 1 [impure] . entropy/gini
No ⇒ ≈1 [-11-]

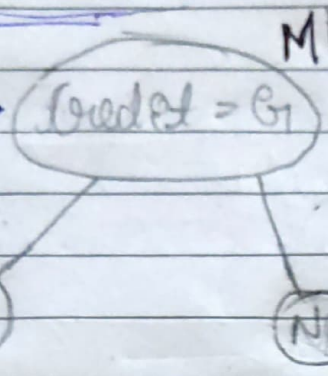
In DT stump-2 → Yes ⇒ 0 [pure split]
No ⇒ 0.. [less impure split]
← good ← tells pure leaf node easily

* we select best Decision tree stump using entropy. ← step1 overview

Step 2: Sum of the total error and performance of stump.

Decision tree which is selected using entropy in step 1

Credit is 'good' the o/p is all 'Yes'



Error

Credit is not good but we have 1 Yes o/p.

Helps to calculate sum of total error

Salary	Credit	Approved	Sample weights
<50K	B	No	1/7
<=50K	G	Yes	1/7
<=50K	G	Yes	1/7
>=50K	B	No	1/7
>50K	G	Yes	1/7
>50K	N	Yes	1/7
<=50K	N	No	1/7

Error

assign equal weights

only 1 wrong record

① Total error = Sum up all the Sample weights = 1/7

② performance of stump $= \frac{1}{2} \log_e \left[\frac{1 - TE}{TE} \right]$ (Total error)

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

$= \frac{1}{2} \ln [6]$

≈ 0.896

indicates weight

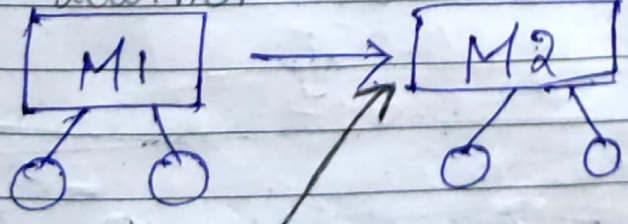
what does it indicates? → α

WORKOT

$f = \alpha_1(M_1) + \alpha_2((M_2) + \cdots + \alpha_n(M_n)$

$\alpha_1 = 0.869$ ⇒ weight

* weak learner [decision tree stump]



wrongly predicted records

overview of step 2

* we calculate performance of stump using Total error.
tells weight to assign to stump

*aim: To send incorrectly ... to next DT stump so we have an ... probability of selecting by updating weight

Step 3: Update the weights for correctly and incorrectly classified points.

* Correctly classified → decrease weight, decrease selecting probability
* Incorrectly classified → increase weight

we have to

because of this there is a high probability of selecting wrong record by next DT stump.

Sample weights	Updated weights
1/7	0.058
1/7	0.058
1/7	0.058
1/7	0.058
1/7	0.058
1/7	0.349
1/7	0.058
	0.697

decreased

increased

incorrectly classified

>50	N	Yes	1/7

* For correctly classified points → weights * $e^{-\text{Performance of stump}}$

$= 1/7 * e^{-(0.896)}$

$= 0.058$

* For incorrectly classified points → weights * $e^{\text{performance of stump}}$

$= 1/7 * e^{0.896}$

$= 0.349$

Step 4: Normalized weights computation and assigning bars.

Normalized why & weight

* we can notice that updated weights sum is not equal to 1

observation step3 → Σ updated weights $\neq 1$ ← 0.697

but initially when we assign the weight

step2 initially → Σ sample weights $= 1$

So, we have to make Σ updated weight $= 1$

also we use to normalize the weight

① updated weights | Normalized weights | Bins assignment

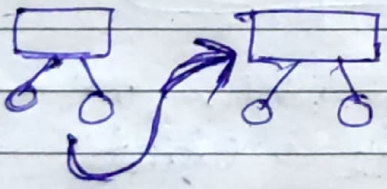
updated weights		Normalized weights	Bins assignment
0.058	% 0.697	0.08	0 - 0.08
0.058		0.08	0.08 - 0.16
0.058		0.08	0.16 - 0.24
0.058		0.08	0.24 - 0.32
0.058		0.08	0.32 - 0.40
0.349		0.50	0.40 - 0.90
0.058		0.08	0.90 - 0.98
$\Sigma = 0.697$		$\Sigma = 1$	

Weight / Σweights → 0.058 / 0.697

Max Bin size

To make

② bins assignment

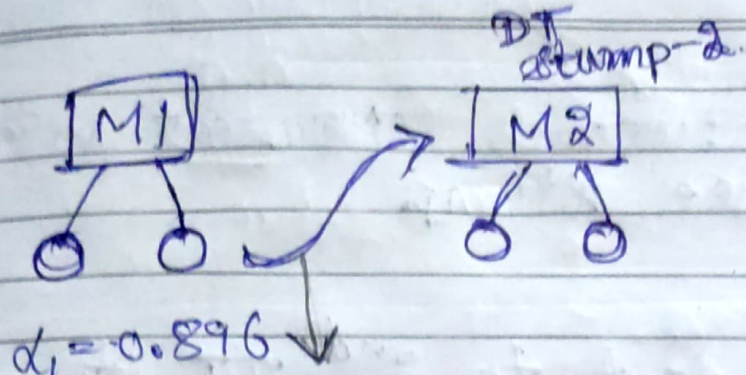


How to make our ml algorithm only send "incorrectly classified points" ?

↓

Ans: we "assign bins"

⇒ Now, we will send the incorrectly classified record frequently to next DT our (DT stump-2)



we have prepare which datapoint/record to be sent

Step 5: Select data points to send to next stump

① Iterative process selecting random value between 0 and 1.

detected records for next stump.

Bank assign	Random number [0 - 1]	&	Credit	Approval
	0.50	→	>50K	N	Yes
	0.10	→	<=50K	G	Yes
	0.60	→	>50K	N	yes
	0.75	→	>50K	N	yes
	0.24	→	<=50K	G	yes
0.4-0.9	0.32	→	>50K	B	No.
	0.87	→	>50K	N	Yes.

randomly generated

check box and select record

observation: incorrectly classified record is picked again and again as "box size" is more as its weight is increased.

which contain more
& correctly classified
records.

* Now, "selected records" are sent to next Decision tree stump.

Step 6: Record sent to 2nd DT stump.

& / Credit / Approval

↓

again follow
Repeat same steps

⇒ TF then

performance stump ⇒ 0.65

$F_1 = \alpha_1 (M_1) + \alpha_2 (M_2)$

$\alpha_1 = 0.856$; $\alpha_2 = 0.65$

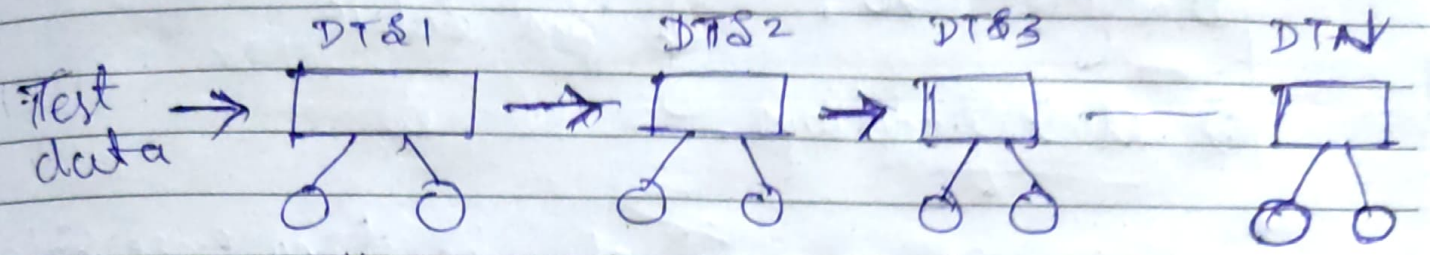
Step 7: Final prediction

Here, for classification problem

* If classification, we use 'mean square error' in place of "entropy"

and at final prediction instead of we get continuous value

Test - data (≤ 50K, G1)



Boosting we will not use voting classifier consider

Yes → Yes | No | Yes — — No

$\alpha_1 = 0.896$ performance of stump | $\alpha_2 = 0.65$ | $\alpha_3 = 0.24$ — $\alpha_n = -0.30$ [denotes weight]

$$\therefore f = \alpha_1(M_1) + \alpha_2(M_2) + \text{— — —} + \alpha_n(M_n)$$

$$\Rightarrow (0.896)(\text{Yes}) + (0.650)(\text{No}) + (0.24)(\text{Yes}) + (-0.30)(\text{No})$$

$$\Rightarrow [(1.136)](\text{Yes}) + [(0.35)](\text{No})$$

Yes > No

performance of day (Yes) = 1.136

performance of day (No) = 0.35

Final output → Yes

* Gradient Boosting algorithm

① Regression. } Can solve
② classification. }

→ it is a part of boosting ensemble technique. where in we create decision tree sequentially to get a "strong learner".

Consider, Regression dataset

Exp	Degree	Salary (y)	ŷ	(y − ŷ) residual R_1
2	B.E	50K	75K	= −25K
3	Masters	70K	75K	= −5K
5	Masters	80K	75K	= +5K
6	PHD	100K	75K	= +25K
		μ = 75K		

independent — dependent

Continuous data → regression problem

Steps:

① Create a Base-model — model should be unbaised [and give default value]

average = 75K

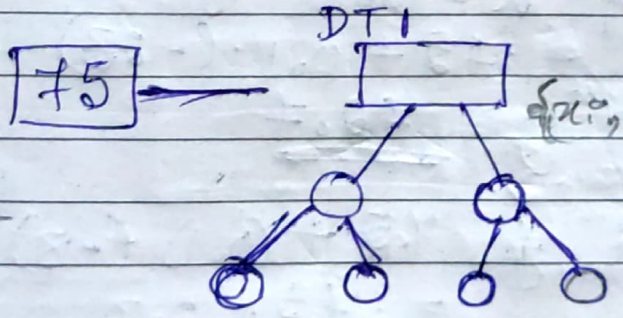
model: 75K → For any input we get 75K as output

② Compute residuals, Errors calculation

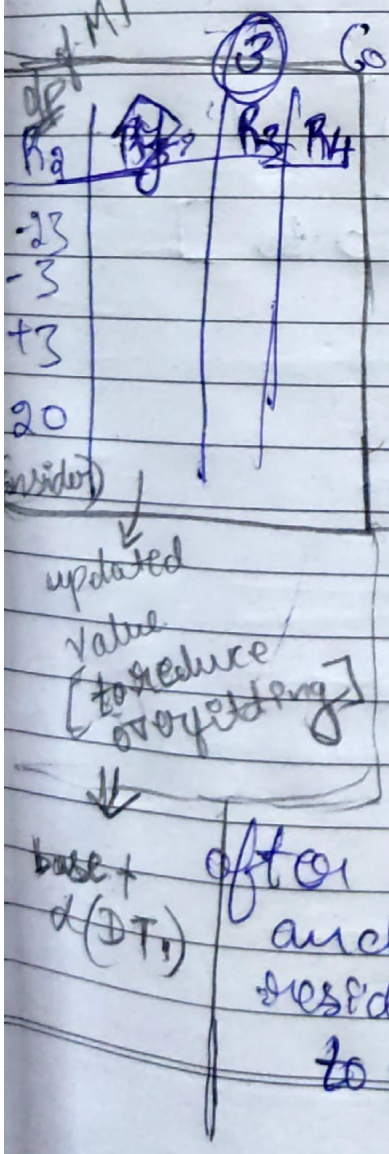
[actual - predicted]

③ Construct of Decision Tree "not stump"

Consider i/p "x_i" and o/p "R_1"



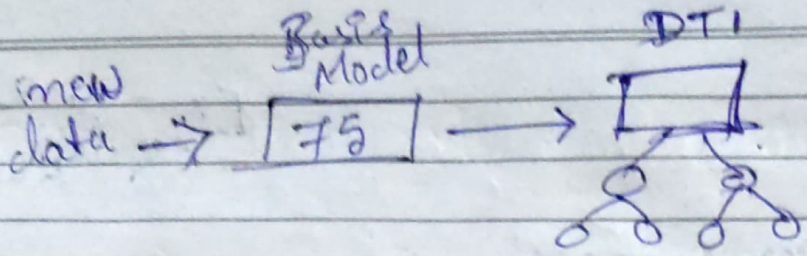
output of the decision tree is "R_2"



after we go and Compute residuals i.e to step2

Consider "R_2"

-23
-3
+3
+20



* predicted o/p ⇒ 75 + (-23) Consider
= 52

↓

Model performing good as
actual value → 50 K
predicted value → 52K

overfitting ←

so to reduce

* Predicted o/p ⇒ 75 + α.(DT1)

α = learning rate [0 to 1]

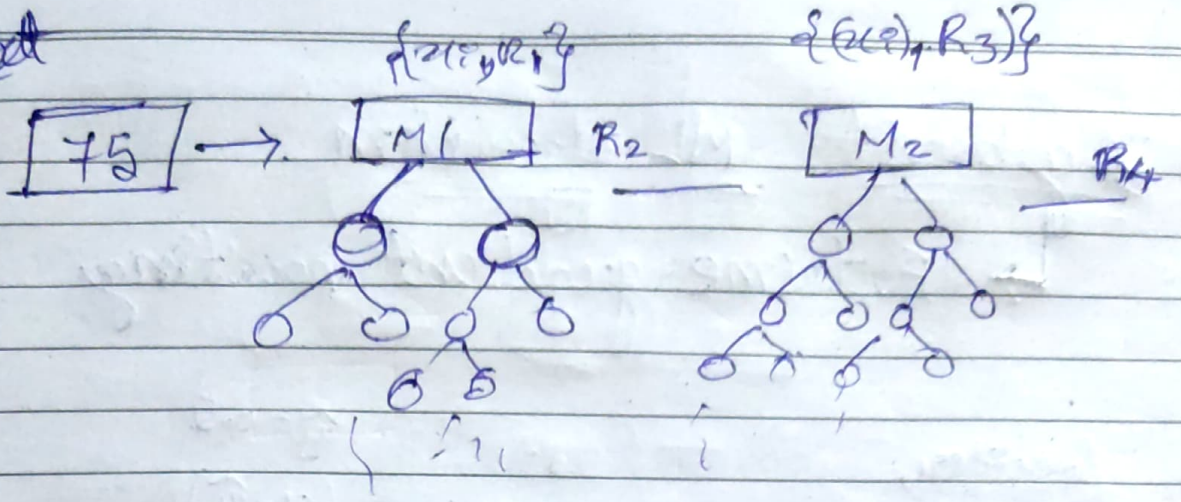
if α = 0.1
= 75 + 0.1 · (-23)
1st row = 72.7

Calculation

2nd row ⇒ 75 + (0.1)(-3)
= 75 - 0.3
= 74.7

y	Rs
72.7	-22.7
74.7	-4.7

Next



* Step. Continues

* Result

$$F(x) = h_0(x) + \alpha_1[h_1(x)] + \alpha_2[h_2(x)] + \cdots + \alpha_n[h_n(x)]$$

learning rate, $\alpha = 0.1$ [For all $\alpha_1, \alpha_2, \cdots \alpha_n$ we can apply]

$$F(x) = \sum_{i=0}^{n} \alpha_i [h_i(x)]$$ ⇒ Final Function

* we can do programming and ——— .

* Xg boost ML algorithm

↳ Extreme gradient boosting

Consider,

Salary	Credit	y Approval	$\hat{y} = 0.5$ $(y - \hat{y})$ (Residual) R_1
<=50K	B	0	-0.5
<=50K	G	1	0.5
<=50	G	1	0.5
>50K	B	0	-0.5
>50K	G	1	0.5
>50K	N	1	0.5
~~>50K~~	~~N~~		
<=50K	N	0	-0.5

Steps

Step 1: Create a base model

← Base model is not based to anything.

↓

$P = 0.5$

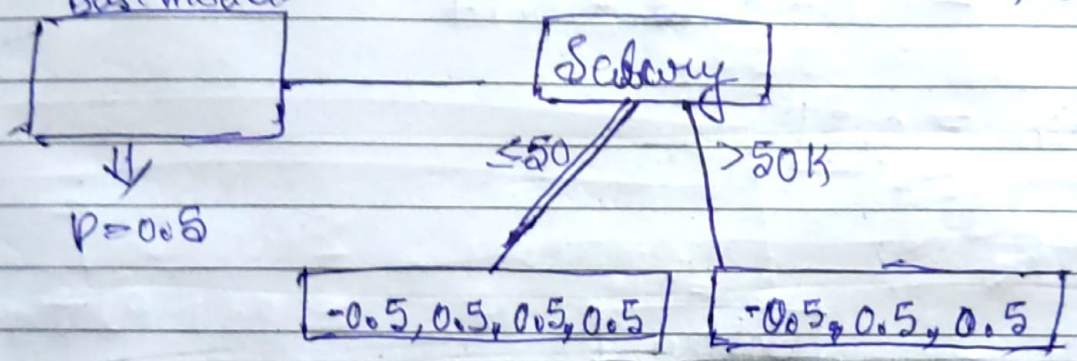
$\hat{y} = 0.5$

Step 2: Construct decision Tree with root node (Salary and Credit) i/p and residual R_1 is o/p

Calculate similarity weight → Calculate gain

step 3 and step 4 tells which feature to be selected as root

[-0.5, 0.5, 0.5, 0.5, -0.5, 0.5, 0.5, -0.5]



Similarity weight [LC] = $\frac{(\sum \text{Residual})^2}{\sum p(1-p)}$ ← left child ← base learner probability

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

$$= \frac{0}{1} = 0$$

Similarity weight [LC] = 0

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

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

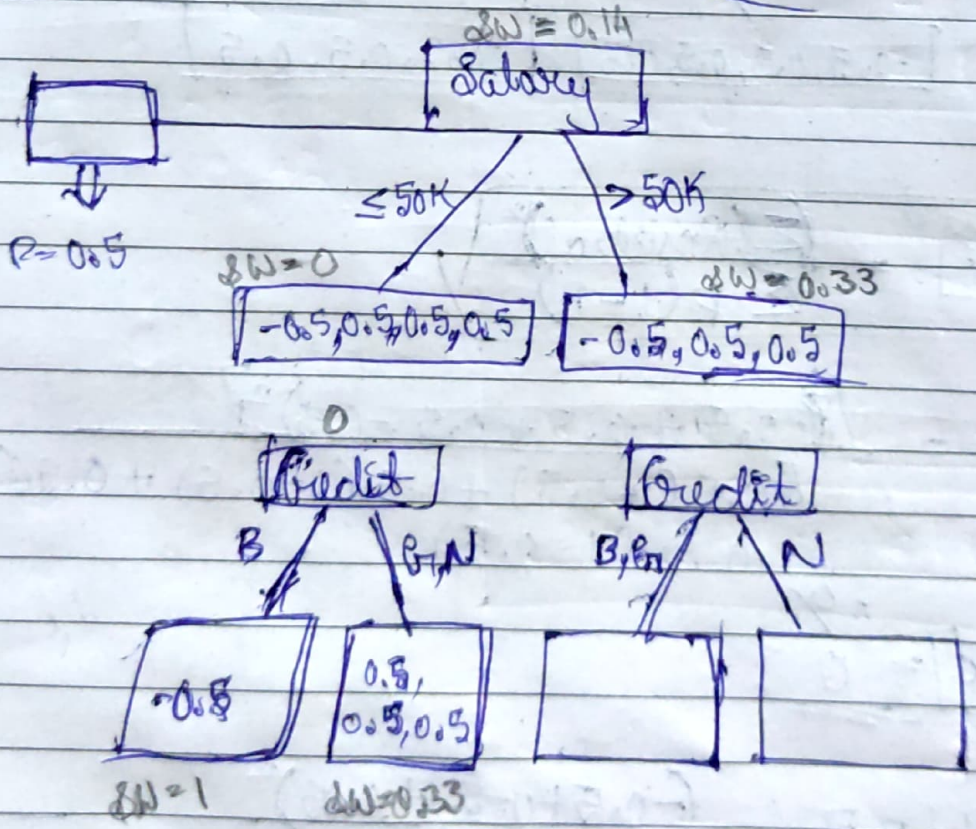
Similarity weight [RC] = 0.33

* similarity weight [root] $= \frac{[-0.5+0.5+0.5-0.5-0.5+0.5+0.5]}{1.75} = \frac{0.25}{1.75} = \frac{1}{7} = 0.14$

* gain = left. + weight - root
child child node

$= 0 + 0.33 - 0.14$

gain $= 0.21$



* Similarity [LC] = $\frac{0.25}{0.25} = 1$

similarity [RC] = $\frac{0.25}{0.75} = \frac{1}{3} = 0.33$

Gain → $1 + 0.33 - 0 = 1.33$

* Similarly, other child:-

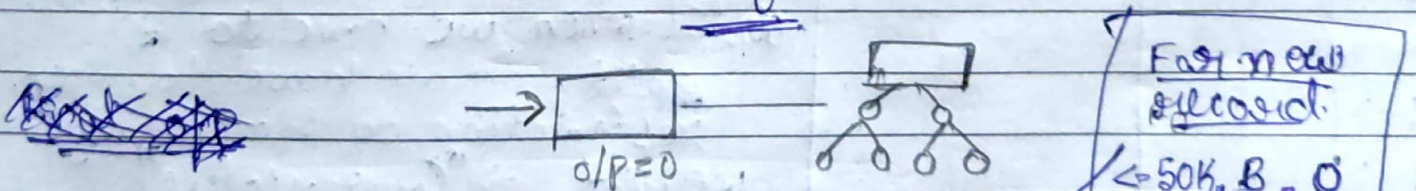
* Now, For any new data

other passes through base model.

$$\log(\text{odds}) = \log\left(\frac{p}{1-p}\right)$$ ← Base model

$$\Rightarrow \log(\text{odds}) = \log\left(\frac{0.5}{1-0.5}\right)$$

$$= 0$$



① New Test Data = $\sigma(0 + \alpha(1))$

- σ → sigmoid activation Function → logistic function
- 0 → base model o/p
- α → learning rate
- 1 → similarity weight from DT for record if $\alpha = 0.1$

$$= \sigma(0.1)$$

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

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

↓

y

2nd record

② [<=50, Gr]

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

If $\alpha = 0.1$

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

$$= \sigma(0.033)$$

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

$y = 0.5$ (base model) for 2nd record $o/p = 0.508$

A	R_1	$\hat{y}$	R_2
y			
0	a	0.52	-0.5
1	a	0.58	0.42
1	a		
0	a		
1	a		
1	a		
0	a		

$y - \hat{y}$ (residual), $y - \hat{y}$

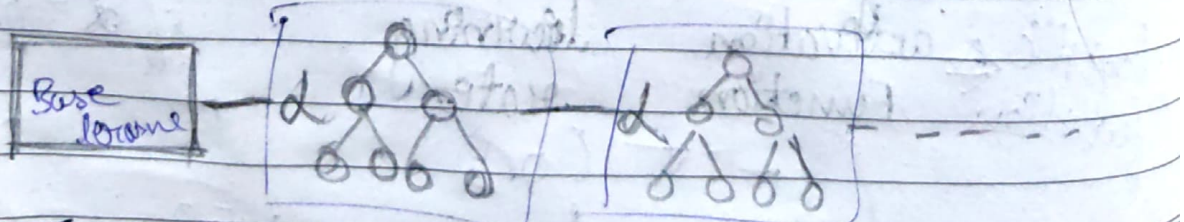
first model o/p

$$\text{similarity weight} = \frac{\Sigma(\text{Residual})^2}{\Sigma P(1-P) + \lambda}$$

λ → hyperparameter

Until when we have to split do we have to keep restriction to similarity weight. → "Cover Value"

Cover Value → $\Sigma P(1-P)$



$$o/p = \sigma[\text{Base learner} + \alpha_1(DT_1) + \alpha_2(DT_2) + \dots + \alpha_n(DT_n)]$$