# ENSEMBLE METHODS / LEARNING

* ensemble method use multiple base(or) learning algorithms to give better performance

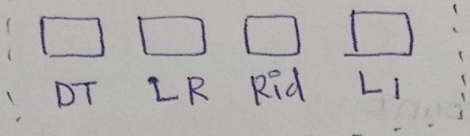
OR

ensemble learning signifies the collection of multiple machine learning algorithms and combine them to form a good model is called as ensemble technique.

Imp;

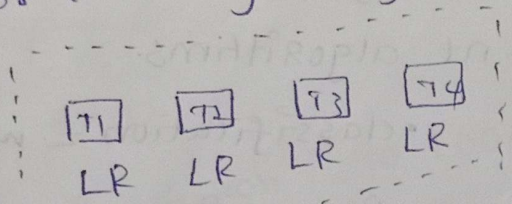
1) All the basc models should be different, this can be done in two types

All the algorithms are different




↳ ensemble learning

same algorith, but giving different data for training every model

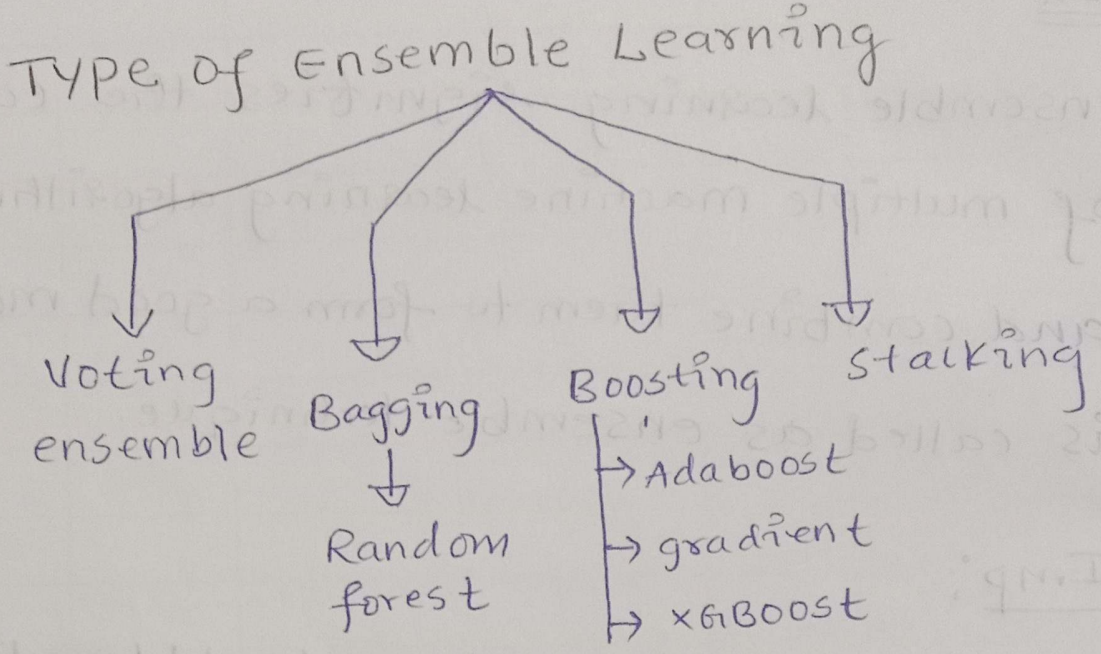



↳ ensemble method.

T1, T2, T3, T4 are different training datasets.

* For classification: The ensemble model gives the output based on majority count.

* For Regression: The out is based on the mean of all the outputs of every algorithm.

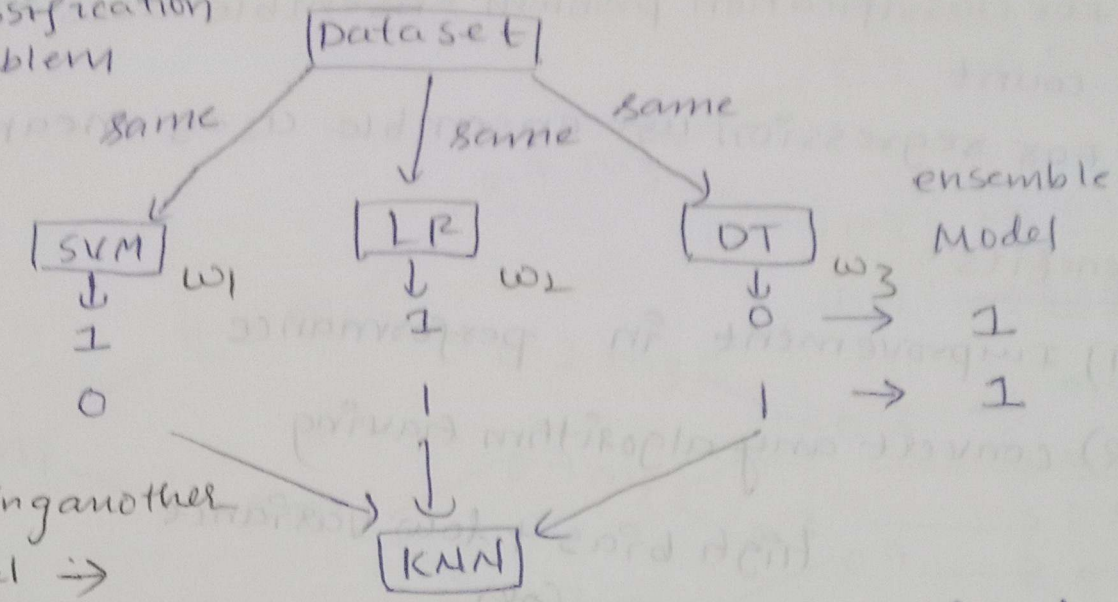



* The above types are existed because of two (or) 2 types of ensemble techniques (which are dicussed in previous page)

* Voting :- Based on ensemble model having different algorithms.
  - classification = majority count
  - Regression = Mean.

* Stacking :- Based on different ml algorithms

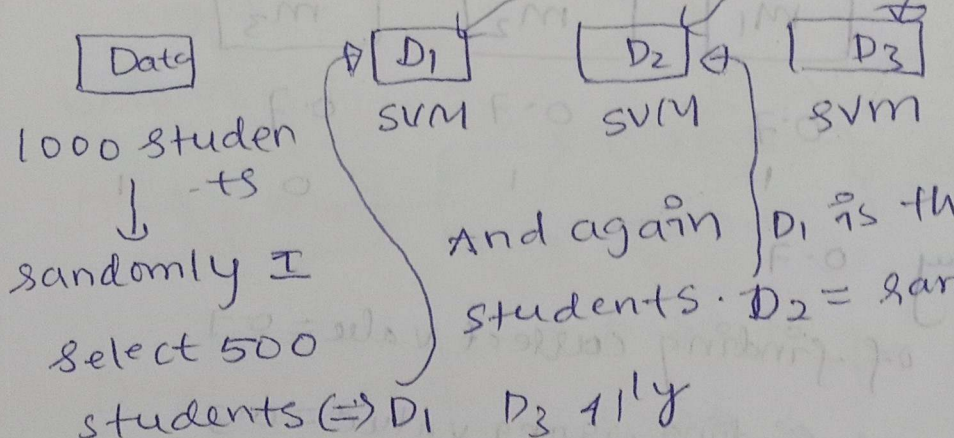
Eg:- classification problem




☆ The KNN will train on the output or predicted values of different algorithms and assign weights based on that data. so that we can clo conclude either SVM, LR, or DT is performing well in the ensemble model.

☆ Both voting and stacking are same but in stacking we additionally only about performance of each algorithm w.r.t ensemble model output by using another ml algorithm.

Bagging :- Bootstraped aggeregation (Based on (same algorithm)



* For classification problem ensemble uses majority count
* For regression use ensemble uses mean.

Benefits:

1) Improvement in performance
2) convert any algorithm having

high bias + low variance
(or)
low bias + high variance
↓
Low bias + low variance

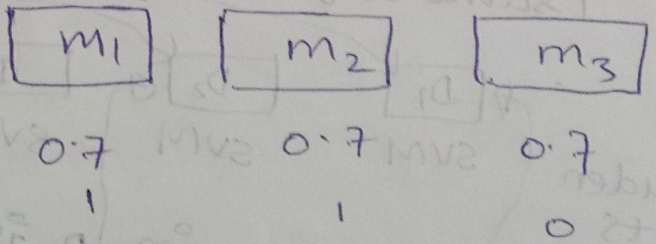
3) Robustness (?)

When to use :- Always

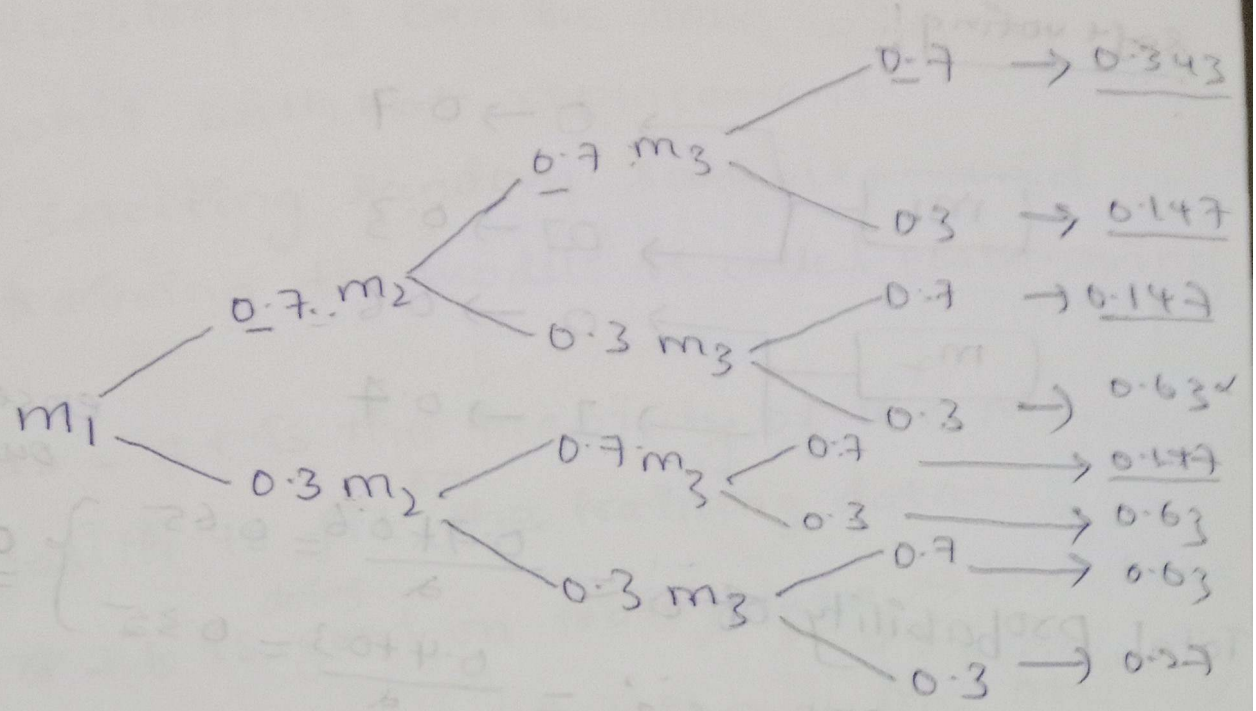
# 1. VOTING:

## 1.1 ⇒ Intution:-

problem:- If we have three different algorithms having accuracy each 0.7




* The accuracy 0.7
* propability of finding correct value = 0.7
  propability of finding wrong value = 0.3

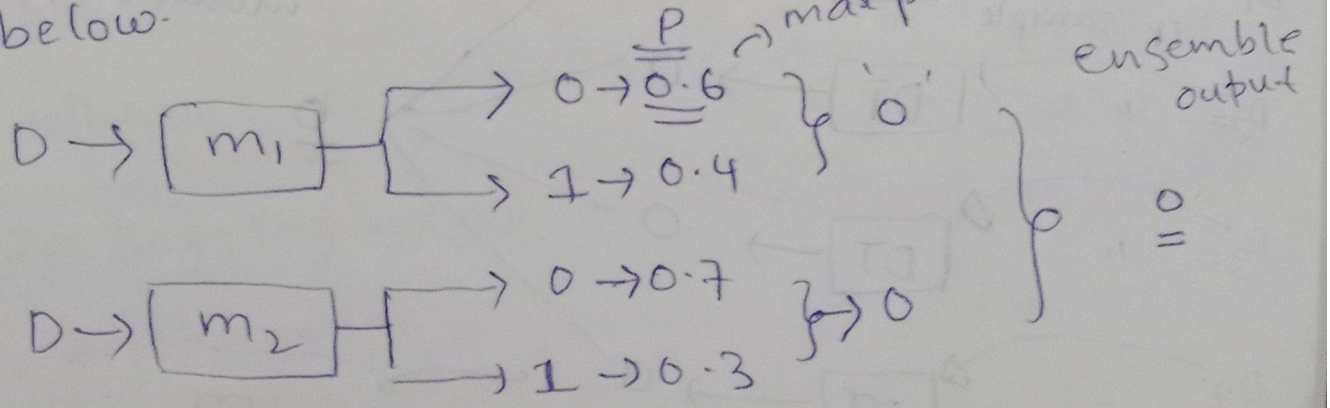



probability of atleast two correct answers by three algorithms = 0.78

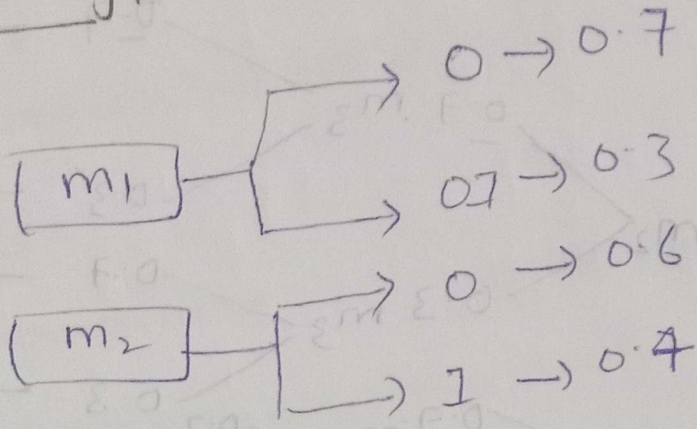
## Voting classifier :-

- soft voting
- hard voting

* hard voting is nothing but choosing the highest probability of the algorithm as shown below.



Soft voting:




ensemble output

Total probability of '0' = $\frac{0.7+0.6}{2} = 0.65$

Total probability of '1' = $\frac{0.4+0.3}{2} = 0.35$

} 0

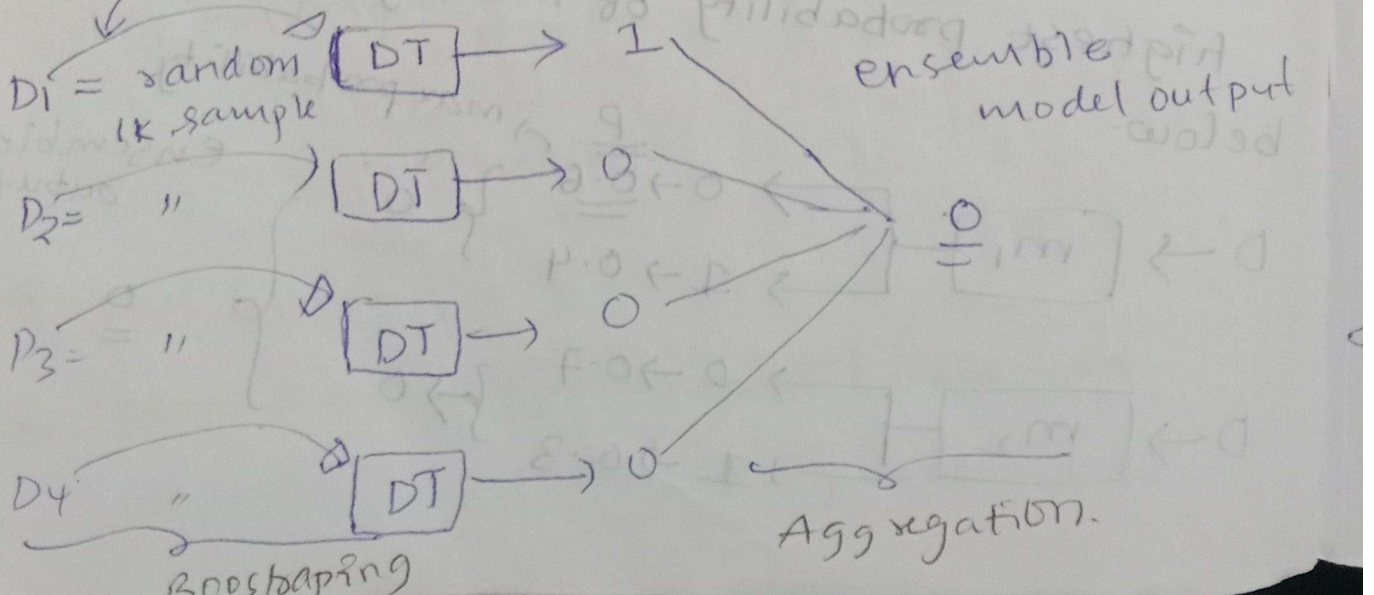
# BAGGING:-

Boostraping

[From a given population selecting random samples]

Aggregation

Dataset = 10K samples



* Boostraping can be done with or with without replacement.
* selecting random samples and training the models is called bootstraping

Bias :- It is the inability of the ML algorith to fit on the training data
* It is doing good on traing data ⇒ Low bia

Variance :- It is the inability of the ML algorithm to fit on test data.

* we use bagging so that the algorithm should maintain low bias and low variance

For ex: generally DT, SVM, KNN are LB A HV type.

* Bagging gives consistent results

pasting :- pasting is nothing but bagging but here we use row sampling without replacement

* In Bagging we do row sampling with replacement

Random Subspaces :- Here we do column sampling with or with out replacement.

Random patches :- Here we do row-sampling and column-sampling with or without replacement.

Random Forest :-

- collection of trees
- In Bagging we do sampling (randomly)

Bagging Vs Random Forest :-

1. In bagging we can select any base estimator like DT, logstic regression, KNN, SVM .. or any. But in randomforest we by default the estimator is Decision tree

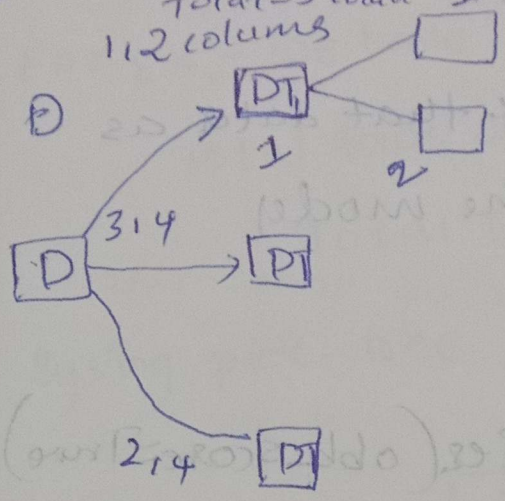
2. This difference is based on feature sampling.
→ In bagging only tree level column sampling will happens:
& In bagging & RT we we are using same DT its bagging has no difference with RF

## Bagging

feature selection = 2

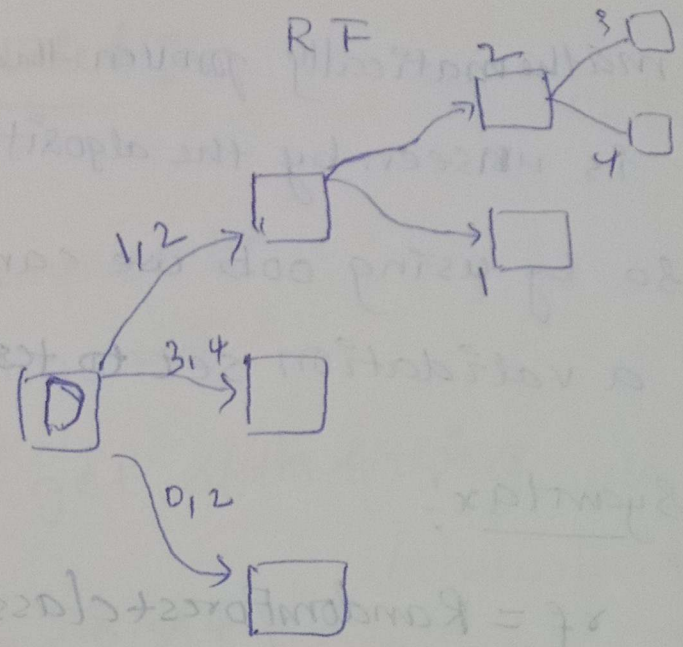
Total = 5 colun

1,2 columns




Here the decision tree forms nodes only on 1,2 column when we consider DT1

## RF




Here at each decision tree we can observe other colums also, however we given only 1,2 columns.

⭐ Random forest do feature selection is node level column sampling.

eg.

*) Due to this randoness of the random forest will be more

## OOB SCORE :- out of bag

or

Evaluation :- bootstrap = True (with replacement of rows).

* Generally when we are using Bagging or Random forest algorithm, with replacement

mathematically proven that 37% of the data is unseen by the algorithms.

*So by using oob we can use that data as a validation set to test the model

Syantax:

```
rf = RandomForestClassifier(obb_score=True)
rf.fit(X_train,ytrain)
rf.obb_score_
```

output = 0.785 (For example)

Feature Importance:- Finding the importance of the column w.r.t Random forest or any algorithm

☆ In this way we can reduce the columns and also feature importance gives interpretability

For example:- In a classification problem whether loan will get sanctioned or not.

* For someone whose loan is not sanctioned he reachout to me and asks why?
* Then as I know the important columns I can interpret them.

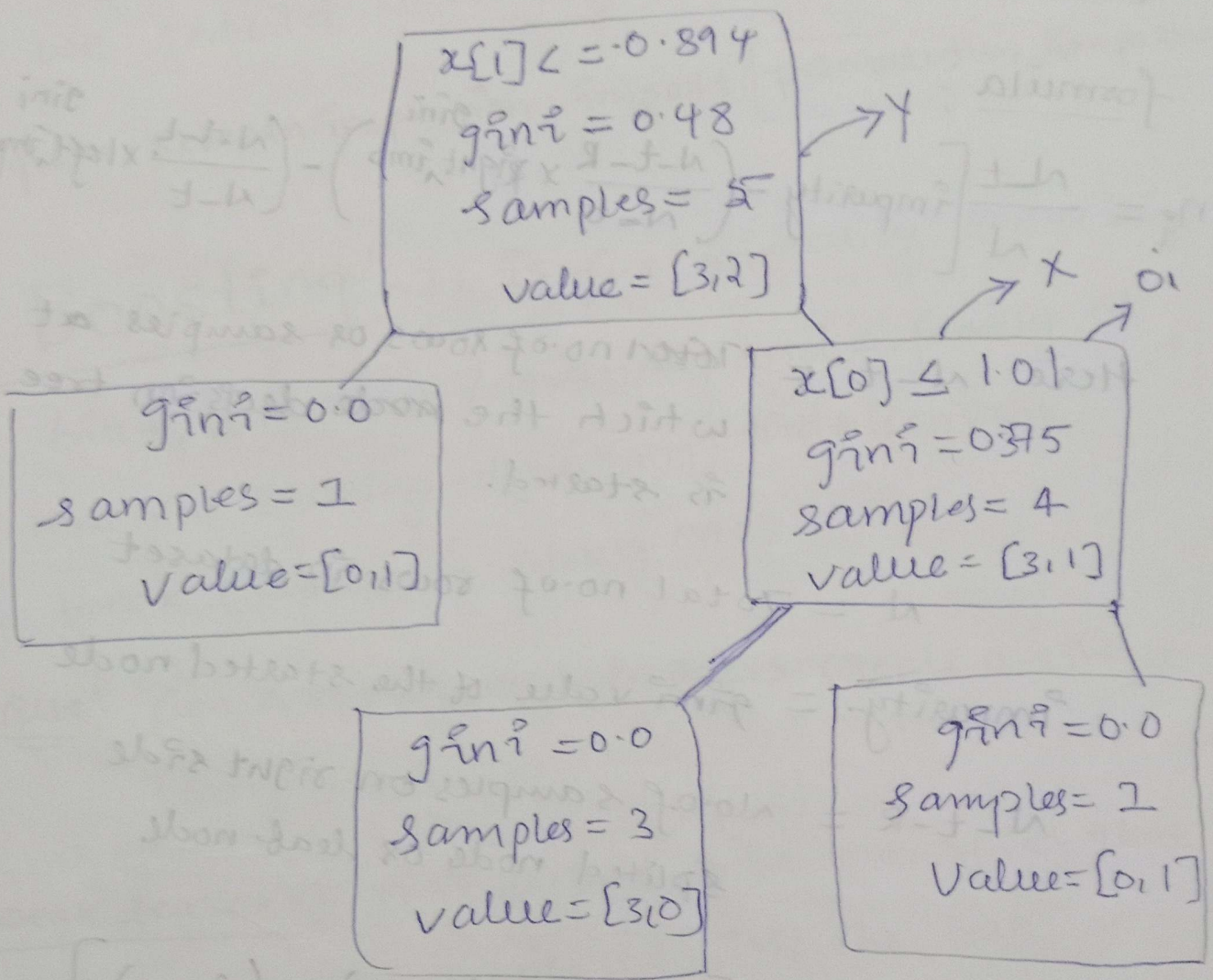
## How feature is calculated :-

For ex :- we have

Total rows or samples = 5

Total columns = 2

By using plot-tree, we get after fitting




Column importance of $0^{th}$ column

$$= \frac{\text{No. of nodes that are splited on } 0^{th}}{\text{Total no. of nodes}}$$

$$= \frac{\text{Importance of nodes that are splited on } 0^{th}}{\text{Total importance of nodes}}$$

For 0th = $\frac{x_i}{x_i + y_i}$ [For our example $x_i = 1, y = 1$, $y_i = 1$ ...]

1st = $\frac{y_i}{x_i + y_i}$

Now how feature importance of nodes are calculated.

formula:

$$n_i = \frac{N_t}{N}\left[\text{impurity} - \left(\frac{N_t - R}{N_t} \times \text{right}_{\text{gini imp}}\right) - \left(\frac{N_t - L}{N_t} \times \text{left}_{\text{gini imp}}\right)\right]$$

Here $N_t$ = no. of rows or samples at which the decision tree is started.

$N$ = Total no. of rows in dataset

impurity = gini value of the started node

$N_{t-R}$ = No. of samples on right side splited node or leaf node

* $(n)_{0th} = \frac{4}{5}\left[0.375 - \left(\frac{1}{4} \times 0\right) - \left(\frac{3}{4} \times 0\right)\right]$

$X = (n)_{0th} = \frac{4}{5} \times 0.375 = 0.3$

* $y = (n)_{1st} = \frac{5}{5}\left(0.48 - \left(\frac{4}{5} \times 0.375\right) - \left(\frac{1}{5} \times 0\right)\right)$

$= (0.48 - 0.3) = 0.18$ //

$$0^{th} = \frac{x}{x+y} = \frac{0.3}{0.3+0.18} = \frac{0.3}{0.48} = 0.625$$

$$1^{st} = \frac{0.18}{0.48} = 0.375$$

* If the data has high cardinality then we should then use

```
from sklearn.inspection import permutation_importan
```

**Boosting**:- It is for primarily reducing bias, and also Variance, and a family of machine learning algorithms that convert weak learners to strong ones.

Que:- Can a set of weak learners create a single strong one?

**weak learner**:- It is defined to be classifier that is only slightly correlated with the true classification. means an algorithm which has accuracy just more than 50%.

(or)

refer to simple models that do only slightly better than Random guessing.

# Adaboost :

1) weak learners

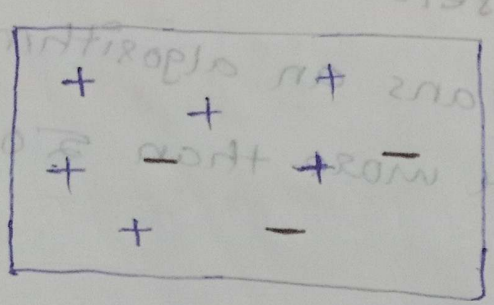
2) Decision stumps

3) -1 and +1 (instead of 0 & 1)

Decision stumps: Actually these are types of week learners. It is a type of decision tree in which its max_depth = 1 [means split happens at one node only.]

* we use decision stumps as our weak learners in the adaboost because we can see what's going on in the algorithm & for good results

* Adaboost is a stage wise additive method (It is formed by multiple weak learners). Here we add weak learners sequentially

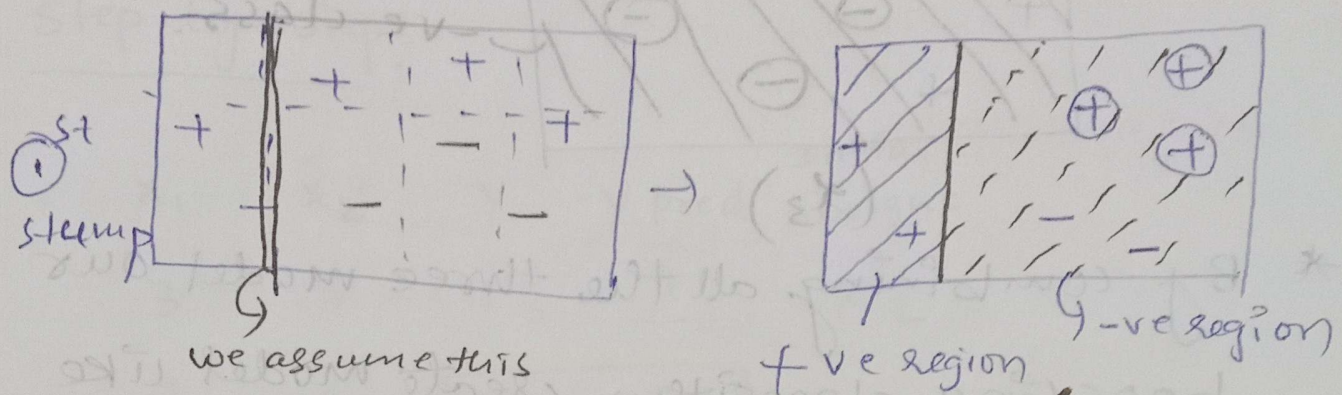


+ → got placement

− → not got

- This is our data set is based on cgpa, iq and output = placement

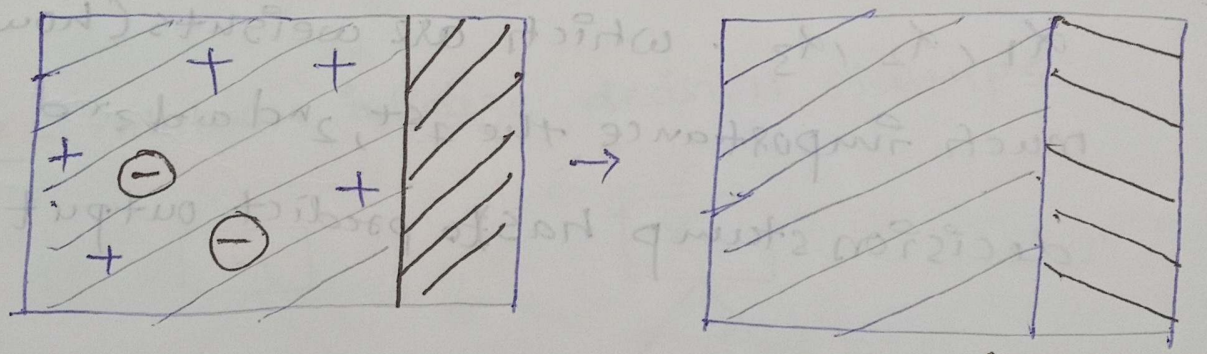
* Now we apply random decision stumps and select one model in such a way that entropy is minimum.




$\alpha_1$

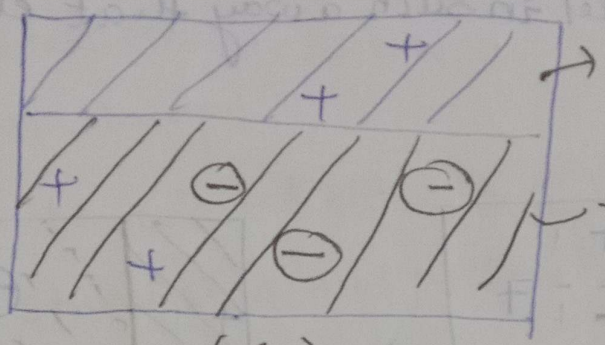
(2) By analyzing above results three ⊕ points are incorrectly classified.

Aim!- Now our 2nd stump aim is create a model in such a way that 3 ⊕ve points are correctly classified. This Aim is also called as **upsampling**



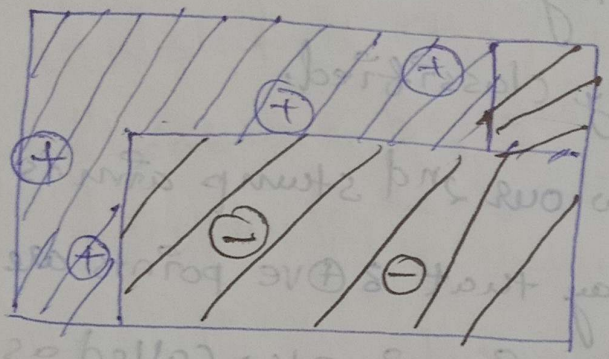

$\alpha_2$

3rd stump!- Aim!- It should create a model in such a way that the 2 ⊖ points should correctly classified.




$(\alpha_3)$

* By combining all the three model our boosting algogoritam create model like this



* In the same process we also calculate $\alpha_1, \alpha_2, \alpha_3$. which are weights (how much importance the 1st, 2nd and 3rd decision stump has to predict output).

Adaboost prediction =

$$\text{sign}\left(\alpha_1 \times h_1(x) + \alpha_2 \times h_2(x) + \alpha_3 \times h_3(x)\right)$$

$$\begin{cases} \text{if } \sum_{i=1}^{n} \alpha_i h_i(x) > 0 \text{ then } \Rightarrow +1 \\ \text{if } \sum_{i=1}^{n} \alpha_i h_i(x) < 0 \text{ then } \Rightarrow -1 \end{cases}$$

here $h_1(x), h_2(x), \ldots$ are the decision stumps.

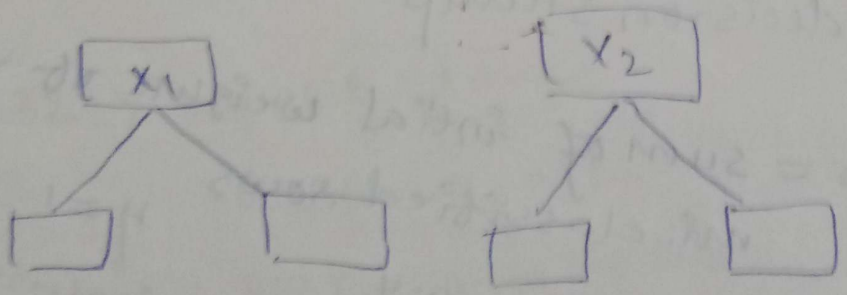
Step-by-step:

| $x_1$ | $x_2$ | Y | Y_Pred | Intial weight | |
|---|---|---|---|---|---|
| 3 | 7 | 1 | 1 | 0.2 | |
| 2 | 9 | 0 | 01 | 0.2 | } mis classified |
| 1 | 4 | 1 | 0 | 0.2 | |
| 9 | 8 | 0 | 0 | 0.2 | |
| 3 | 7 | 0 | 0 | 0.2 | |

Step-1: our adabost model intialize weights to each and every row by using formula $= \frac{1}{\text{no of rows}}$

In above case $w = \frac{1}{5} = 0.2$

Step-2: Adaboost intialize decision stumps




stump-1 stump-2

* In the above two stumps which has less entropy or gini that decision stump (weak learner is selected).

Step-3 : Now we predict

Aim: The main aim of adaboost is to assign weights to each and every decision stump(tree)

* By using one decision stump we calculated the y-pred values

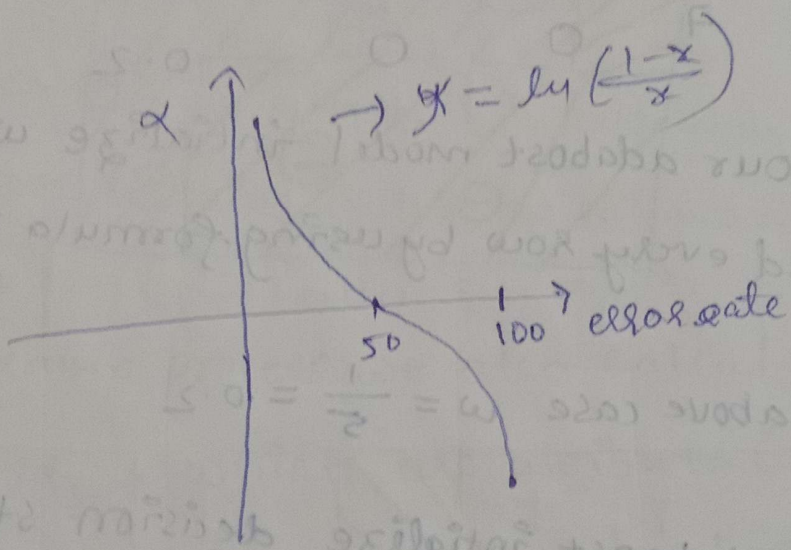
$\eta$ = learning rate.

$$\eta\alpha = \frac{1}{2} ln\left(\frac{1-error}{error}\right)$$

Ex: If error rate = 0% the $\alpha$ is max

error " = 100% the $\alpha$ is min

" " = 50% $\alpha = 0$




$M_1$ (1st decision stump)

error = sum of intial weights of the misclassified rows

$= 0.2 + 0.2 = 0.4$

$\eta = 1$ learning rate

$$\eta\alpha_1 = \frac{1}{2} ln\left(\frac{0.6}{0.4}\right) = \frac{1}{2} ln\left(\frac{3}{2}\right)$$

$$= \frac{1}{2}(0.477 - 0.301) = \frac{0.176}{2}$$

$w_2 = 0.69$

$w_3 = 1.09$

$\alpha_1 = \frac{0.405}{a} = \boxed{0.20 = \alpha_1}$

Aim:- To upsample misclassified weights.

For misclassified:-

$$\text{new\_wt} = \text{curr\_wt} \times e^{\alpha_1} = 0.2 \times e^{0.2} = 0.24$$

For correctly classified:-

$$\text{new\_wt} = \text{curren\_wt} \times e^{-\alpha_1} = 0.2 \times e^{-0.2} = 0.16$$

* Aim: To get the sum of updated weights is =1 as intial weights we normalize them by dividing each one with Total sum

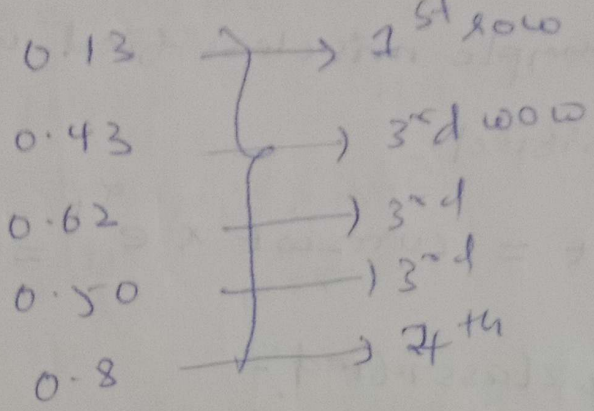
| updated | normalized |
|---|---|
| 0.16 | 0.166 |
| 0.24 | 0.25 |
| 0.24 | 0.25 |
| 0.16 | 0.166 |
| 0.16 | 0.166 |
| 0.96 | 1 |

Now our data set looks like this:

| $x_1$ | $x_2$ | y | newwt | range |
|---|---|---|---|---|
| | | | 0.166 | 0 - 0.166 |
| | | | 0.25 | 0.166 - 0.416 |
| | | | 0.25 | 0.416 - 0.666 |
| | | | 0.166 | 0.666 - 0.832 |
| | | | 0.166 | 0.832 - 1.0 |

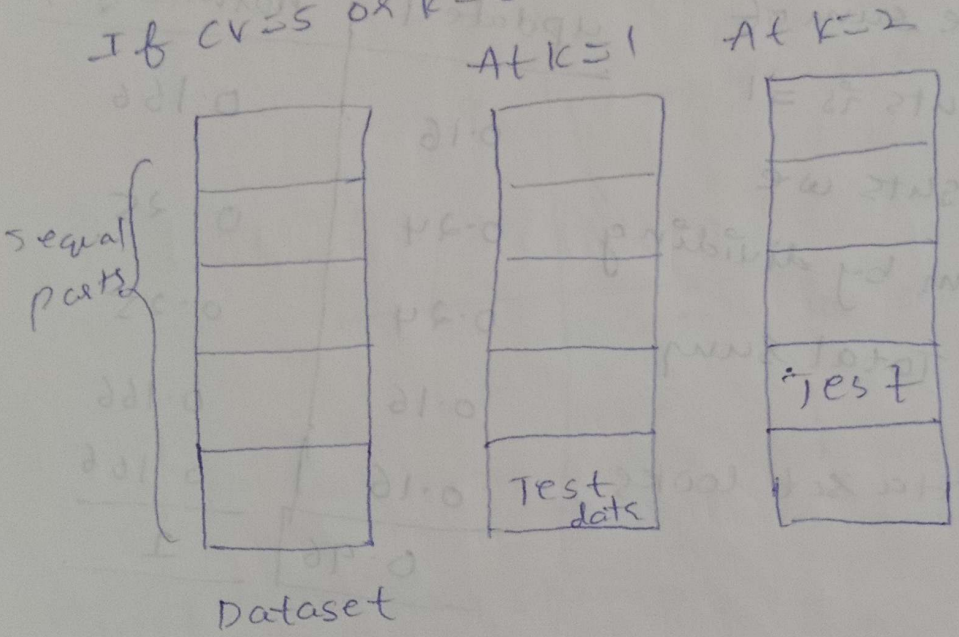
* Now adaboost select 5 random numbers from 0 to 1

For eg - these are those numbers :-




Cross_validation :- or K-fold CV :-

If CV=5 or K=5




ly.

learning rate hyperparameter ($\eta$) :-

If $\eta$ is small then there is a chance of of underfitting because the upsampling weights are not have much difference with weights of correctly classified.

# Bagging Vs Boosting :-

1. Type of model :-

→ In bagging we use a model which has "Low Bias & High variance" eg: fully grown decision tree.

→ In boosting we use model which has "High variance & Hig bias" Low

eg :- Decision stump.

2. sequential learning vs parallel :-

→ bagging → models will train parallely at a time

→ boosting → model will train one after the other in sequential order.

3. weightage of base learners :-

→ Bagging = In bagging all the base model will get or have equal priority means equal weights.

→ Boosting = In boosting every model will get assigned by different weights