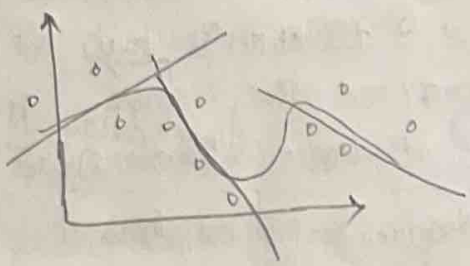
Unit-5 : Local Models.

Introduction:

Divide i/p space into local regions & learn simple models in each patch.



→ unsupervised : Competitive, online clustering

→ supervised : Radial-basis function, mixture of experts.

Competitive Learning:

- Competitive learning is used because it is as if k-groups, or rather the units representing these groups, compete among themselves to be one responsible for representing an instance.
- model is also called winner-take-all, it is as if one group wins and gets updated and the others are not updated at all.

Adaptive Resonance Theory: (ART)

- is a type of NN developed by Stephen Grossberg & gail carpenter.
- uses unsupervised learning technique in 1987.
  Adaptive - open to new learning.
  Resonance - without discarding previous or old info.
- Implement a clustering alg.
- Alg. checks whether it fits into one of already stored clusters.

↳ If it fits then i/p is added to cluster.
↳ Otherwise new cluster is formed.

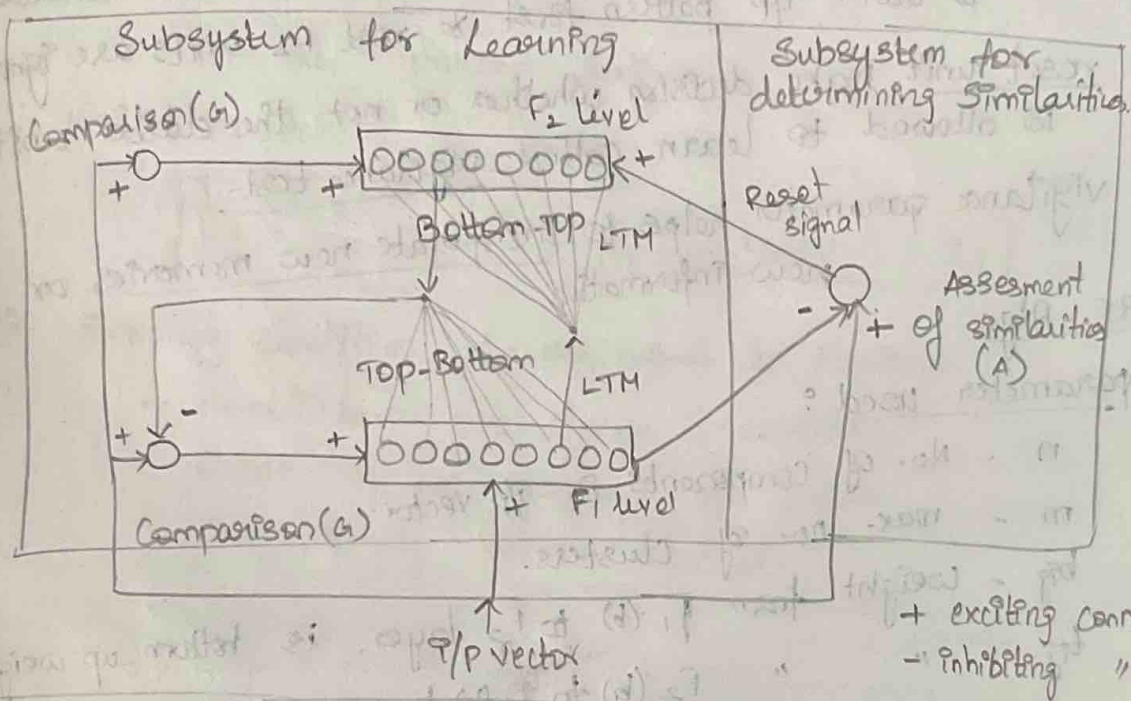
Types of ART:

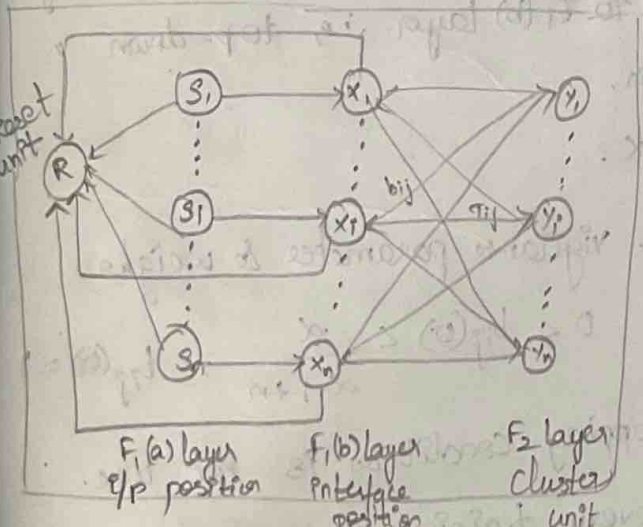
1) ART1 (Capable of clustering binary i/p values)
2) ART 2 (" " " continuous-valued i/p data)
3) Fuzzy ART (fuzzy logic)
4) ART MAP (ART learns based on previous ART module)
5) FART MAP. (supervised ART with fuzzy logic)

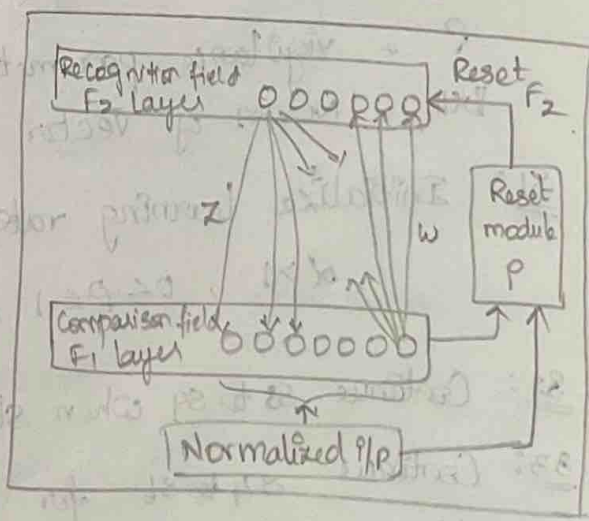
ART Application:

→ Target Recognition → Mobile Control Robot
→ Medical Diagnosis → Signature verification.

ART Architecture:







Basic ART Structure

- Basic ART is unsupervised in nature.
- F_1 layer or comparision field. (i/p are processed)
- F_2 layer or recognition field (consists of clustering units)
- reset module (acts as control mechanism)

ART process:

→ F_1 layer accepts i/p & perform some processing & transfer it to F_2.

→ there exist a set of weighted interconnection for controlling the degree of similarity b/w units in F_1 & F_2 layer.

→ F_2 layer is competitive layer.
Cluster unit with large net i/p becomes the candidate to learn i/p pattern first & rest F_2 units are ignored

→ reset unit make decision whether or not the cluster unit is allowed to learn called vigilance test.

→ vigilance parameter helps to incorporate new memories or new information.

ART Alg:

parameter used:

n - No. of components in i/p vector.
m - max. no. of clusters.
b_{ij} - weight from F_1 (b) to F_2 layer, i.e bottom-up weights
t_{ij} - " " F_2 to F_1 (b) layer. i.e, top-down "
ρ = vigilance parameter.
$\|x\|$ = Norm of vector x.

S1: Initialize learning rate, vigilance parameter & weights.

$$\alpha > 1, \quad 0 < \rho \leq 1, \quad 0 < b_{ij}(0) < \frac{\alpha}{\alpha - 1 + n}, \quad t_{ij}(0) = 1$$

S2: Continue S3 to S9 when stopping condition is not true.

S3: Continue S4 to S6 for every training i/p.

S4: set activation of all $F_1(a)$ & F_1 units.

$F_2 = 0$, $F_1 a$ = i/p vectors.

S5: i/p signal from F_1 a to F_1 b layer must be sent like

$$s_i = x_i$$

S6: for every inhibited F_2 node,

$$y_j = \sum_i b_{ij} x_i \quad , \quad y_j \neq -1.$$

S7: perform S8 to S10 when reset is true.

S8: find J for $y_J \geq y_j$ for all nodes j

S9: Calculate activation on F_1 b $\quad x_i = s_i t_{Ji}$

S10: Check reset condition

if $\|x\| / \|s\| <$ vigilance parameter ρ,

then inhibit node J & go to S7.

else if $\|x\| / \|s\| \geq \rho$,

then proceed further.

S11: weight updating for node J.

$$b_{ij}(\text{new}) = \frac{\alpha x_i}{\alpha - 1 + \|x\|} \quad , \quad t_{ij}(\text{new}) = x_i$$

S12: Stopping condition for alg. must be checked.

- do not have any change in weight.
- Reset is not performed for units.
- max. No. of epochs reached.

Online k-means:

$$E\left(\{m_i\}_{i=1}^{k} \mid X\right) = \frac{1}{2} \sum_t \sum_i b_i^t \|x^t - m_i\|^2$$

where $b_i^t = \begin{cases} 1, & \text{if } \|x^t - m_i\| = \min \|x^t - m_i\| \\ 0, & \text{otherwise.} \end{cases}$

$X = \{x^t\}_t$ is sample , $b_i^t = 1$ if m_i is closest center.

m_i, $i = 1, \ldots k$ are cluster centers.

k-means updates the centers as,

$$m_i = \frac{\sum_t b_i^t x^t}{\sum_t b_i^t}$$

- online k-means is obtained by doing stochastic gradient descent.

error for single instance is.

$$E^t(\{m_i\}_{i=1}^k | x^t) = \frac{1}{2}\sum_i b_i^t \|x^t - m_i\|^2 = \frac{1}{2}\sum_i \sum_{j=1}^{d} b_i^t (x_j^t - m_{ij})^2$$

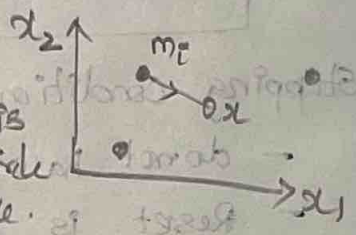
update rule for each instance x^t:

$$\Delta m_{ij} = -\eta \frac{\partial E^t}{\partial m_{ij}} = \eta\, b_i^t (x_j^t - m_{ij})$$

- for convergence, η is gradually decreased to 0, which implies stability-plasticity dilemma.
- if η is large, m_i may oscillate.

* Competitive n/w can be implemented as one-layer recurrent n/w, i/p layer contains i/p vector x, no bias unit, values of o/p units are b_i & they are perceptrons:

$$b_i = m_i^T x$$



shaded circle is center & empty circle is i/p instance.

Choose max. of b_i & set it equal to 1.

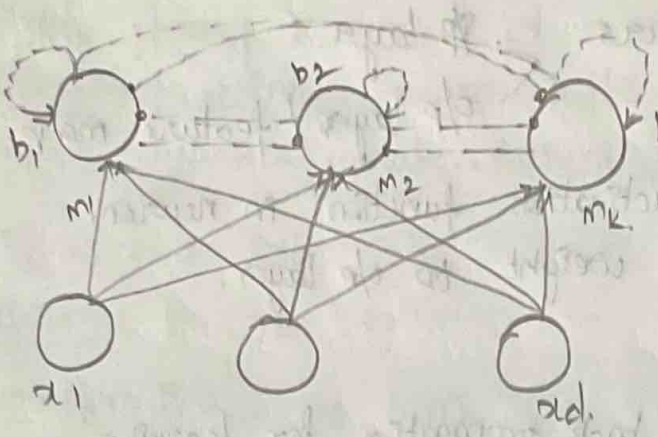
Alg:

Initialize m_i, $i = 1 \ldots k$ for eg. to k random x^t
Repeat
  for all $x^t \in X$ in random order
  $i \leftarrow \arg\min_j \|x^t - m_j\|$
  $m_i \leftarrow m_i + \eta (x^t - m_i)$
Until m_i Converge

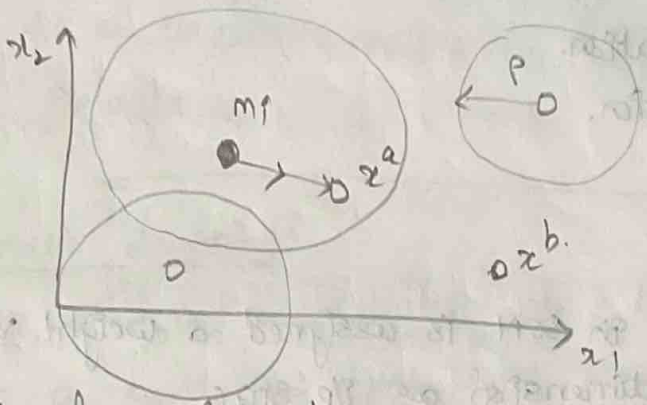
update rule can be rewritten as:

$$\Delta m_{ij}^t = \eta\, b_i^t x_j^t - \eta\, b_i^t m_{ij}$$

where $\Delta m_{ij}^t = \eta\, b_i^t x_j^t$ is hebbian learning.



Winner take all competitive neural n/w, which is a n/w of k perceptrons with recurrent connections at o/p.

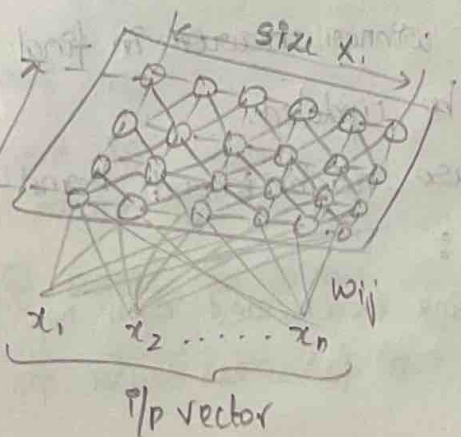


distance from x^a to closest center is less than vigilance value ρ

Self-organizing Maps: (SOM)

- SOM or kohonen's map is a type of ANN introduced by Teuvo kohonen in 1980's.
- Is trained using unsupervised learning.
- uses competitive learning to adjust weights.
- used in dimensionality reduction.
- helps to discover correlation b/w data.

Architecture:



- has 2 layers → i/p layer
  → o/p layer / feature map.
- don't have activation function in neuron.
- directly pass weight to o/p layer.

Training:

- doesn't use back propagation for learning.
- Competetive learning processes:
  - Competetion.
  - Cooperation.
  - Adaptation.

SOM process:

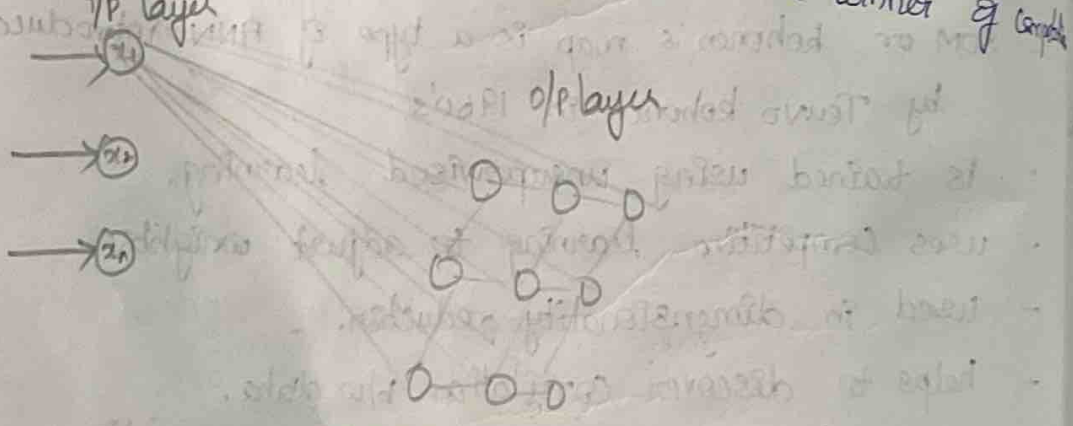
1) Competetion:

S1: each neuron in SOM is assigned a weight vector same dimension as i/p space.

S2: each neuron of o/p layer will have vector with dimension

S3: Compute distance b/w each neuron & i/p data

S4: Neuron with lowest distance will be winner of competition



2) Cooperation:

- update the vector of winner neuron in final process.
- its neighbor also will be updated.
- to choose neighbor use neighborhood kernel function
- function depends on:
  → time (time incremented each new i/p data)
  → distance. (how far is other neuron from winner neuron)

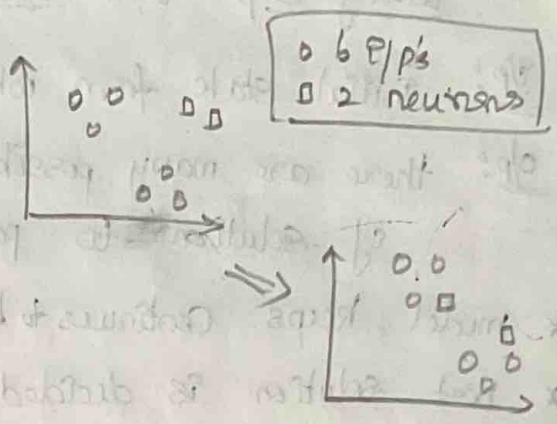
3) Adaptation:

- after choosing winner neuron & its neighbors we compute neurons update. Adjust weight.

Process:

Operations:

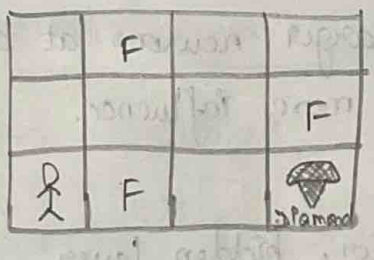
1. Select random i/p.
2. Compute winner neuron.
3. Update neurons.
4. Repeat for all i/p data.
5. Classify i/p data.



Reinforcement learning:

- is about taking suitable action to maximize reward in a particular situation.
- is a supervised learning where training data has the answer key with it so the model is trained with correct answer.
- but in RL, there is no answer, but agent decides what to do to perform the given task.
- in absence of training dataset, it is bound to learn from its experience.

Example:



→ agent / Robot
F → fire
→ Diamond / Reward

- we have agent reward & many hurdles inbetween.
- the agent is supposed to find best possible path to reach the reward.
- Goal of agent is to get reward that is diamond & avoid hurdles that is fire.

- the robot learns by trying all possible path & then choosing the path which gives him reward with least hurdles.
- each Correct step will give robot a reward & each wrong step will substract reward.

I/p: Initial state from which the model start.

O/p: there are many possible o/p as there are variety of solution to particular problem.

* model keeps continues to learns.
* best solution is decided based on max. reward.

Application:
- used in robotics for industrial automation.
- used in ML & data processing.

Radial Basis functions: (RBF)

- is a real-valued function, the value of which depends only on distance from the origin.
- Gaussian function is most common type of radial basis func
- RBF for neuron consists of center & radius (spread).
  - ↳ radius may vary b/w different neurons.
  - ↳ as spread grows larger, neurons at distance from a point have more influence.

RBF n/w Architecture:
- consists of I/p layer, hidden layer, summation layer.

1) I/p layer:
   - → consists of one neuron for every predictor variable.
   - → I/p neuron pass the value to each neuron in the hidden layer.

ii, hidden layer:

- Contains the variable No. of neuron.
- each neuron comprises a RBF centered on point.
- No. of dimensions coincide on No. of predictor variable.
- when 'x' vector of i/p values is fed from i/p layer, a hidden neuron calculates the euclidean distance b/w the test case & neuron's center point.
- It then applies the kernel function using the spread value
- resulting value gets fed into summation layer.

iii, o/p layer / summation layer:

- value obtained from hidden layer is multiplied by weight related to neuron & passed to summation.

Training the RBFN:

→ training process includes selecting these parameters:

- prototype (mu)
- beta coefficient for every RBF neuron.
- matrix of o/p weights b/w neurons & o/p nodes.

can be trained using gradient descent

Advantages:

→ good generalization
→ faster training.
→ only 1 hidden layer.

Bagging And Boosting:

- Ensemble learning helps improve / allows the production of better predictive performance compared to single model
- Types of ensemble learning:
  - ↳ Bagging:
  - ↳ Boosting:

Bagging:

- Bootstrap Aggregating, also known as bagging is a ML ensemble meta-alg. designed to improve the stability & accuracy of ML alg.
- decreases the variance & helps to avoid overfitting.
- Special case for model averaging approach.

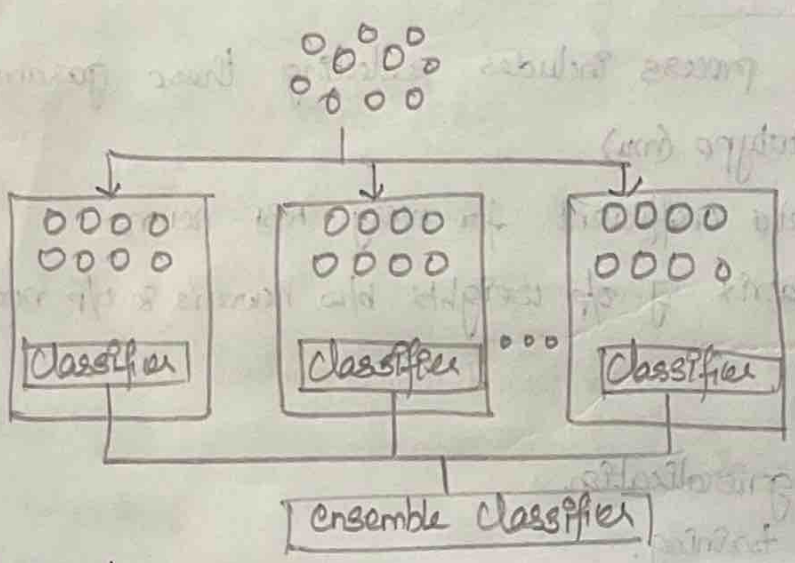
Steps:

S1 - multiple subsets are created from original dataset with equal tuples, selecting observation with replacement.

S2 - A base model is created on each of these subsets.

S3 - Each model is learned in parallel with each training set & independent of each other.

S4 - final predictions are determined by combining the predictions from all models.



Eg: Random forest.

Boosting:

- is an ensemble modeling technique that attempts to build a strong classifier from No. of weak classifiers.
- first, model is built from training data.
  then 2nd model is built which tries to correct the errors present in 1st model.
- this procedure is continued & model are added until either complete training dataset is predicted correctly

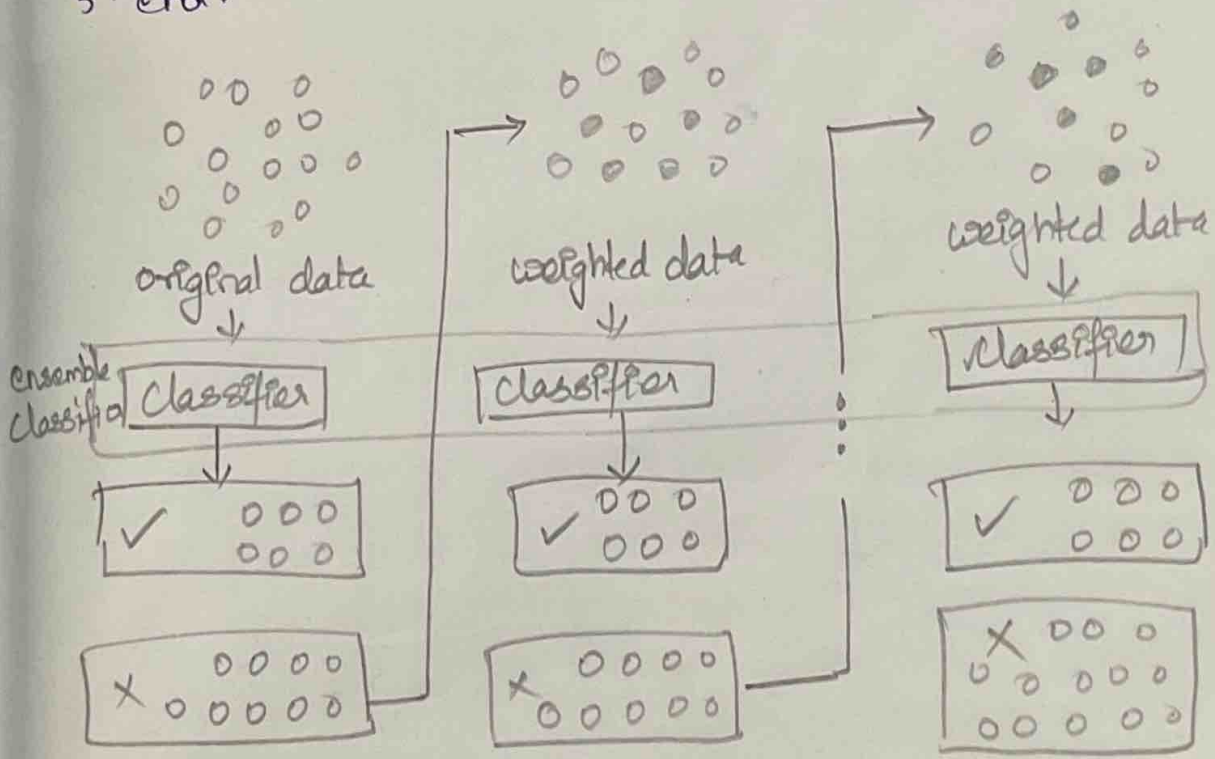
or max. No. of models is added.

Algorithm:

- Adaboost, an adaptive boosting alg. is successful alg. developed for purpose of binary classification.
- papular boosting technique that combines multiple "weak classifiers" into single "strong classifier".

Steps:

1. Initialise dataset & assign equal weight to each of data point.
2. Provide this an i/p to model & identify the wrongly classified data points.
3. Increase the weight of wrongly classified data points & decrease the weight of correctly classified data point. Normalize the weight of all data points.
4. If (got required result)
   goto s5
   else
   goto s2.
5. end.



Boosting