Module 2: Supervised Learning

* **Preference bias:** The inductive bias of ID3 is give a preference for certain hypothesis that can be eventually enumerated. This form of bias is called....

* **Restriction bias:** The bias of CANDIDATE-ELIMINATION algo. is in the form of a categorical restriction on the set of hypothesis considered. This form of bias is called...

Q: Which type of inductive bias is required in order to generalize beyond the training data?

⟹ Preference bias is more desirable than a restriction bias, because it allows the learner to work within a complete hypothesis space that is assured to contain the unknown target function.

★ ISSUES IN DECISION TREE LEARNING:

(a)

⟹ ~~Avoiding~~ Overfitting the Data

Definitation: Given a hypothesis space H, a hypothesis $h \in H$ is said to **overfit** the training data if there exists some alternative hypothesis $h' \in H$, such that h has smaller error than h' over the training data, but h' has a smaller error than h over the entire distribution of instances.

⟹ There are several approaches to avoiding overfitting in decision tree learning.

(i) Approaches that stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data.

(ii) Approaches that allow the tree to overfit the data, and then post-prune the tree.

Note:- Although the first of these approaches might seem more direct, the second approach of post-pruning overfit trees has been found to be more successful in practice. This is due to difficulty in the first approach of estimating precisely when to stop growing the tree.

→ (i) Reduced Error Pruning:-

two different sets like

⟹ We use training set and validation set to prevent overfitting.

step 1:- Reduced Error Pruning is to consider each of the decision nodes in the tree to be candidates for pruning.

step 2:-

Note:-

Draw

step 2:- Pruning a decision node consists of removing the subtree rooted at that node, making it a leaf node, and assigning it the most common classification of the training examples affiliated with that node.

Note:- Nodes are removed only if the resulting pruned tree performs no worse than the original over the validation set.

⟹ Nodes are pruned iteratively, always choosing the node whose removal most increases the decision tree accuracy over the validation set.

⟹ Pruning of nodes continues until further pruning is harmful (i.e. decreases accuracy of the tree over the validation set).

⟹ In this approach, available data has been split into three subset: 1. The training examples
2. The validation examples
3. The test examples

Drawback:- When the data is limited, withholding part of it for the validation set reduces even further the number of examples available for training.

(ii) Rule Post-Pruning:-

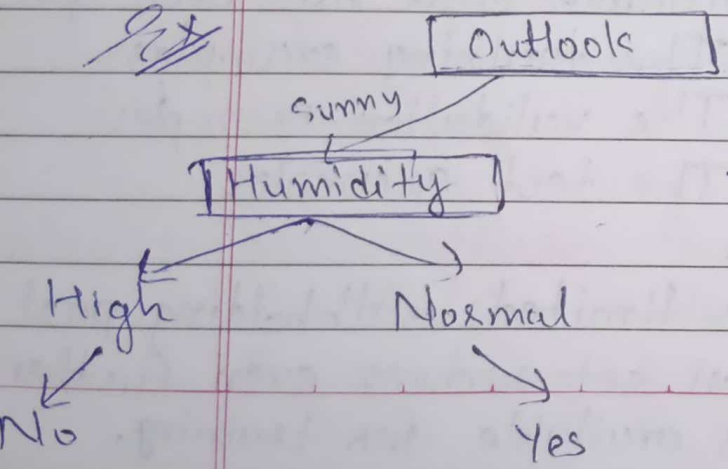
step 1: Infer the decision tree from the training set, growing the tree until the training data is fit as well as possible and allowing overfitting to occur.

step 2: Convert the learned tree into an equivalent set of rules by creating one rule for each path from the root node to a leaf node.

step 3: Prune (generalize) each rule by removing any preconditions that result in improving its estimated accuracy.

step 4: Sort the pruned rules by their estimated accuracy, and consides them in this sequence when classifying subsequent instances.

Ex



Rule:-

IF (Outlook = Sunny) ∧ (Humidity = High)
THEN PlayTennis = No

* Why convert the decision tree to rules before pruning?

⟹ Converting to rules allows distinguishing among the different contexts in which a decision node is used. Because each distinct path through the decision tree node produces a distinct rule, the pruning decision regarding that attribute test can be made differently for each path.

⟹ Converting to rules removes the distinction between attribute tests that occur near the root of the tree and those that occur near the leaves.

⟹ Converting to rules improves readability. Rules are often easier for people to understand.

(b)

⟹ Incorporating Continuous-Valued Attributes

⟹ In intial definition of ID3 is restricted to attributes that take on a discrete set of values.

1. Target attribute whose value is predicted by the learned tree must be discrete valued.

2. The attributes tested in the decision nodes of the tree must also be discrete valued.

⟹ This second restriction can easily be removed so that continuous-valued decision attributes can be incorporated into the learned tree.

⟹ This can be accomplished by dynamically defining new discrete-valued attributes that partition the continuous attribute value into a discrete set of intervals.

⇔

Temperature:	40	48	60	72	80	90
PlayTennis :	No	No	Yes	Yes	Yes	No

step 1:- Pick a threshold, c, that produces the greatest information gain.

⟹ In the current example, there are two candidate thresholds, corresponding to the values of Temperature at which the value of PlayTennis changes:

$\frac{(48+60)}{2} = 54$ and $\frac{(80+90)}{2} = 85$

⇒ The information gain can then be computed for each of the candidate attributes, Temperature$_{>54}$ and Temperature$_{>85}$, and the best can be selected (Temperature$_{>54}$).

⇒ This dynamically created boolean attribute can then compete with the other discrete-valued candidate attributes available for growing the decision tree.

(c) ⇒ Alternative Measures for Selecting Attributes:-

⇒ There is a natural bias in the information gain measure that favors attributes with many values over those with few values.

⇒ But, simply put, it has so many possible values that it is bound to separate the training examples into very small subsets. Because of this it will have a very high information gain relative to the training examples, despite being a very poor predictor of the target function over unseen instances.

Ex Date attribute

Solution: One alternative measure that has been used is **gain ratio**. The gain ratio measure penalizes attributes such as Date by incorporating a term, called **split information**, that is sensitive to how broadly and uniformly the attribute splits the data:

$$\text{SplitInformation}(S, A) \equiv -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

$$\text{GainRatio}(S, A) = \frac{\text{Gain}(S, A)}{\text{SplitInformation}(S, A)}$$

(I) ⇒ Handling Training Examples with Missing Attribute Values:-

⇒ One strategy for dealing with the missing attribute value is to assign it the value that is most common among training examples at node n.

⇒ A second procedure is to assign a probability to each of the possible values of A rather than simply assigning the most common value to A(x).

(E) ⟹ Handling Attributes with Differing Costs:

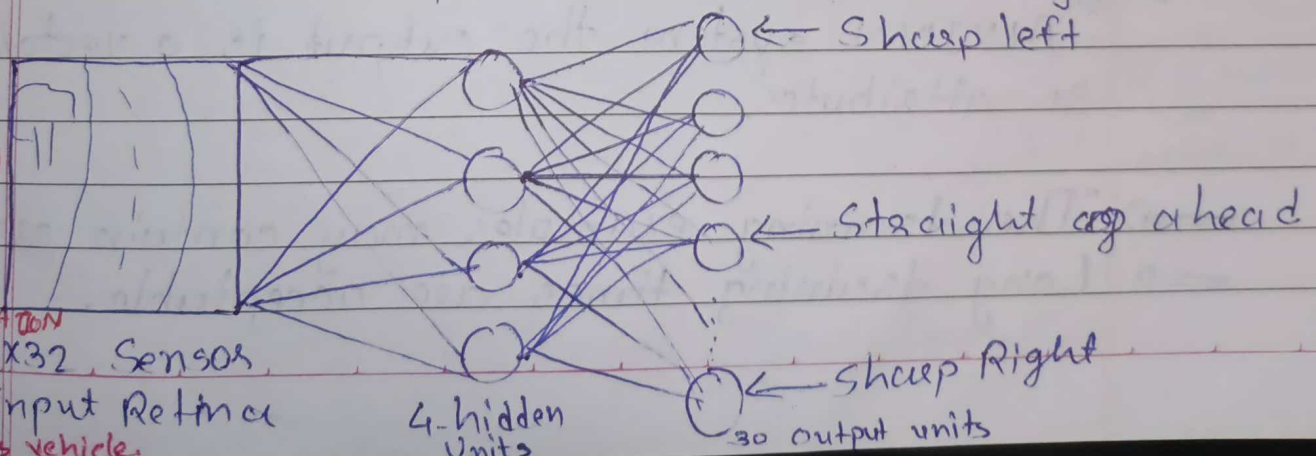
⟹ Measuring attribute cost and prefer cheap cost compare to higher cost attribute. But learner can also use costly/higher cost attribute if it provides good gain value.

⟹ Problem in mention approach is it is bias towards lower cost attribute.

Ex robot & sonar: time required to position
medical diagnosis: cost of a laboratory test

Module: 3: Artificial Neural Network & Genetic Algorithms

⟹ Neural network learning methods provide a robust approach to approximating real-valued, discrete-valued, and vector-valued target functions.



The ALVINN System uses BACKPROPAGATION to learn to steer an autonomous vehicle.

✱ Appropriate Problems for Neural Network Learning:

⟹ ANN learning is well-suited to problems in which the training data corresponds to noisy, complex sensor data, such as inputs from cameras and microphones.

Note:- The BACKPROPAGATION algorithm is the most commonly used ANN learning technique.

✱ characteristics of problems where ANN is used:-

⟹ Instances are represented by many attribute value pairs.

⟹ The target function output may be discrete-valued, real-valued or a vector of several real or discrete valued attributes.

Ex

ALVINN system the output is a vector of 30 attributes.

⟹ The training examples may contain errors.

⟹ Long training times are acceptable.

⟹ Fast evaluation of the learned target function may be required.

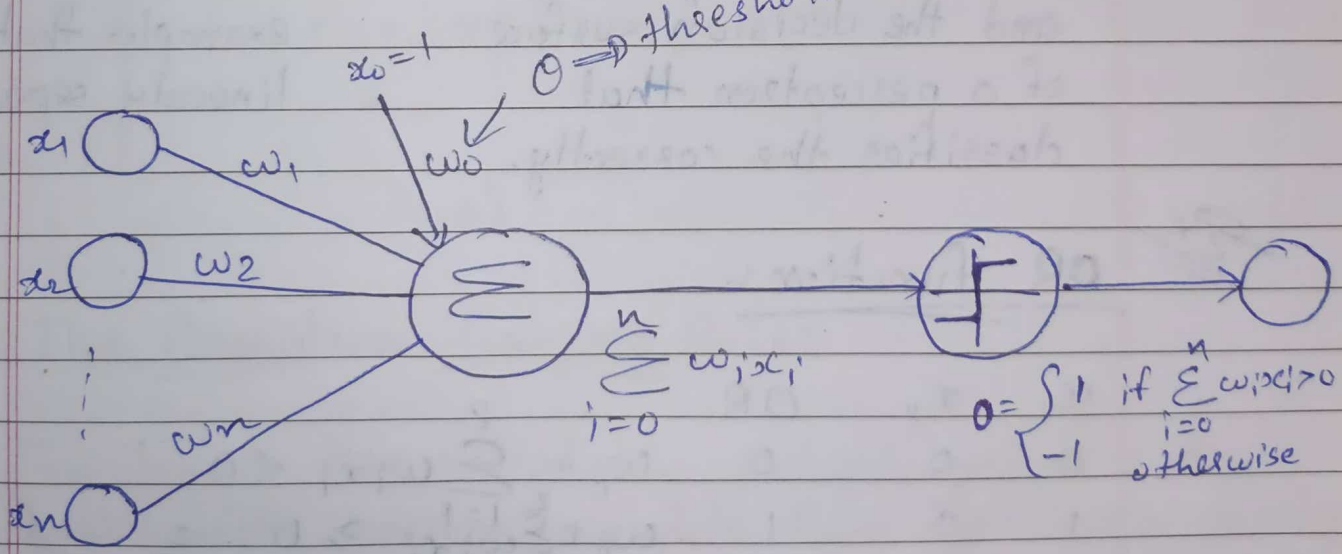
⟹ The ability of humans to understand the learned target function is not important.

★ PERCEPTRONS:-

⟹ One type of ANN system is based on a unit called a perceptron.

⟹ Input: vector of real-valued inputs calculates a linear combination of these i/ps & o/p

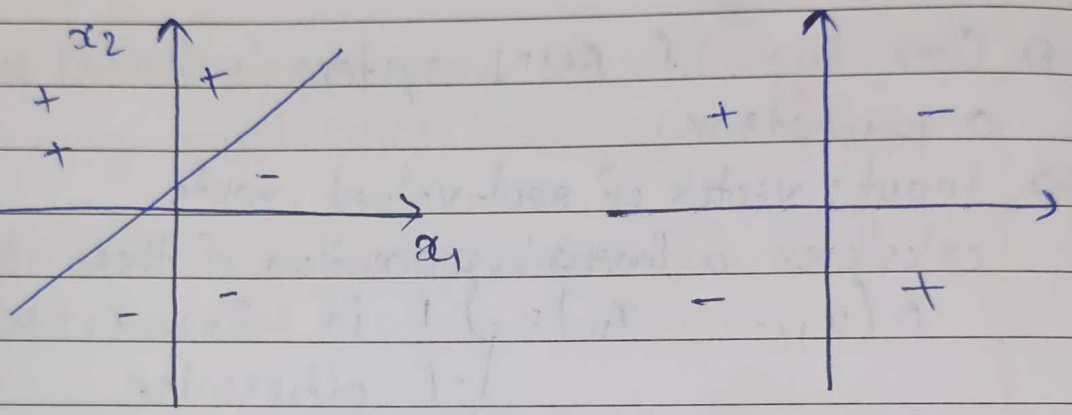
$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > 0 \\ -1 & \text{otherwise} \end{cases}$$



Representation of Perceptron

→ A single perceptron can be used to represent many boolean functions.

⇒ Perceptrons can represent all of the primitive boolean functions AND, OR, NAND (¬AND), and NOR(¬OR).



(a) A set of training examples and the decision surface of a perceptron that classifies the correctly.

(b) A set of training examples that is not linearly separable.

Ex

OR function :-

x_1	x_2	OR
0	0	0
1	0	1
0	1	1
1	1	1

$$w_0 + \sum_{i=1}^{2} w_i x_i < 0$$

$$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$$

$$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$$

$$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$$

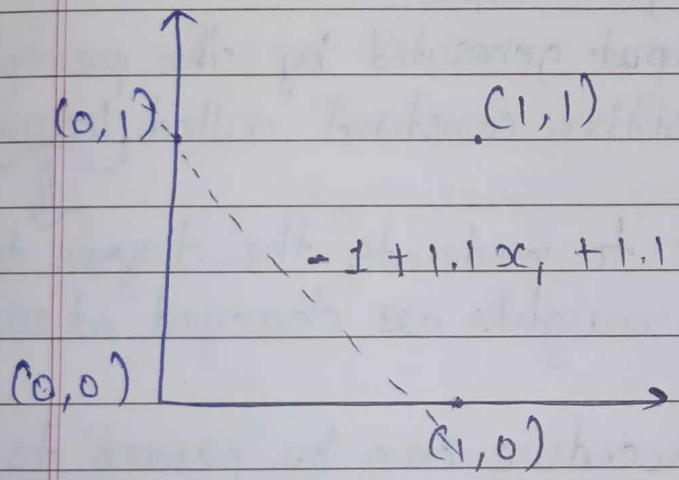
$w_0 + w_1 0 + w_2 0 < 0 \Rightarrow w_0 < 0$

$w_0 + w_1 0 + w_2 \cdot 1 \geq 0 \Rightarrow w_2 \geq -w_0$

$w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 \Rightarrow w_1 \geq -w_0$

$w_0 + w_1 1 + w_2 1 \geq 0 \Rightarrow w_1 + w_2 \geq -w_0$

⟹ One of the possible solution to this set of inequalities is $w_0 = -1$, $w_1 = 1.1$, $w_2 = 1.1$



✱ The Perceptron Training Rule:-

⟹ To learn an acceptable weight vector is to begin with random weights, then iteratively apply the perceptron to each training example, modifying the perceptron weights whenever it misclassifies an example.

→ This process is repeated, iterating through the training examples as many times as needed until the perceptron classifies all training examples correctly.

Perceptron training rule;

$$w_i \leftarrow w_i + \Delta w_i$$

where;

$$\Delta w_i = \eta (t - o) x_i$$

t = target output

o = output generated by the perceptron

η = positive constant called learning rate

to moderate the degree to which weights are changed at each step.

Note:- Above learning procedure can be proven to converge within a finite number of applications of the perceptron training rule to a weight vector that correctly classifies all training examples, provided the training examples are linearly separable and provided a sufficiently small η is used.

→ If the data are not linearly separable, convergence is not assured.

★ Gradient Descent and the Delta Rule:-

⟹ Delta rule is designed to overcome the difficulty faced by perceptron rule. That means if the training examples are not linearly separable, the delta rule converges toward a best-fit approximation to the target concept.

$$\text{training error } E(\vec{w}) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

where;

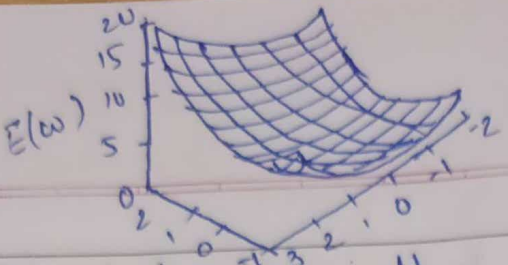
D = set of training examples.

t_d = target output for training example d

o_d = output of the linear unit for training example d.

Note:- Here, E as as a function of $\vec{w}$ because the linear unit output o depends on this weight vector.

⟹ Gradient descent search determines a weight vector that minimizes E by starting with an arbitrary initial weight vector, then repeatedly modifying it in small steps.



⟹ At each step, the weight vector is altered in the direction that produces the steepest descent (slope) along the error surface. This process continues until the global minimum error is reached.

$$\nabla E(\vec{w}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots, \frac{\partial E}{\partial w_n}\right]$$

where;

$\nabla E(\vec{w})$ = vector, whose components are the partial derivatives of E with respect to each of the w_i

training rule for gradient descent --

$$\vec{w} = \vec{w} + \Delta\vec{w}$$

where,

$$\Delta\vec{w} = -\eta \nabla E(\vec{w})$$

η = learning rate determines the step size in the gradient descent search.

Note:- negative sign is present because we want to move the weight vector in the direction the decreases E.

✱ Gradient-Descent algorithm:

GRADIENT-DESCENT(training_examples, η)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, where $\vec{x}$ is the vector of input values, and t is the target output value.

→ Initialize each w_i to some small random value.

→ Until the termination condition is met, Do

(i) Initialize each Δw_i to zero.

(ii) For each $\langle \vec{x}, t \rangle$ in training_examples, D

- Input the instance $\vec{x}$ to the unit and compute the output o
- For each linear unit weight w_i, Do

$$\Delta w_i = \Delta w_i + \eta (t - o) x_i$$

(iii) For each linear unit weight w_i, Do

$$w_i = w_i + \Delta w_i$$

✱ Practical difficulties in applying gradient desent:

⟹ Converging to a local minimum can sometimes be quite slow (i.e. it can require many thousands of gradient descent steps).

⟹ If there are multiple local minima in the error surface, then there is no guarantee that the procedure will find the global minimum.

$$\Delta w_i = \eta (t - o) x_i$$

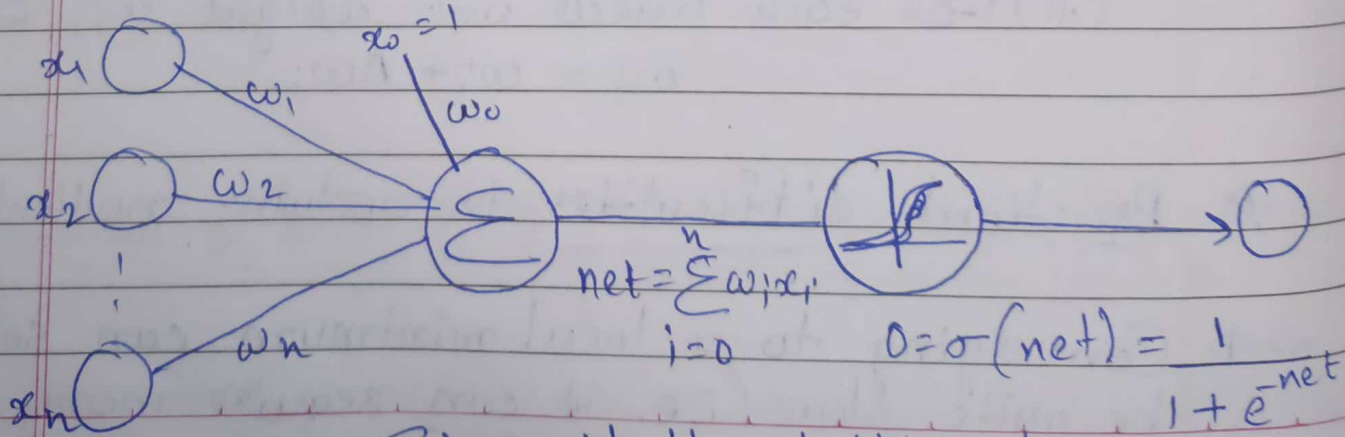
⇓

delta rule / least-mean-square rule.

✡ Multilayer Networks & Backpropagation algo:

⟹ Single perceptrons can only express linear decision surfaces.

⟹ Multilayer networks learned by the Backpropagation algo. are capable of expressing a rich variety of nonlinear decision surfaces.



Sigmoid threshold unit

☆ Unsupervised Learning:-

challenges
- Computational complexity due to a high volume of training data.
- Longer training times
- Human intervention to validate o/p variables.

⇒ Only Inputs
⇒ Clustering.
⇒ K-means clustering.

☆ K-NN algo:- (Lazy learner algorithm)

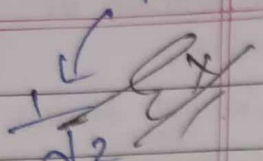
⇒ Most basic instance-based method.
⇒ The nearest neighbors of an instance are defined in terms of the standard euclidean distance.

Algo:-
- step1 Choose the number of neighbors
- step2 Calculate Euclidean distance
- step3 Identify Nearest Neighbors

① Training algo:-
- For each training example $\langle x, f(x) \rangle$ add the example to the list training_examples.

② Classification algo:-
- Given a query instance x_q to be classified
  - Let $x_1, \ldots x_k$ d -

for weighted K-NN



Sr. No.	Height	weight	Target	Distance
1	150	50	Medium	8.06
2	155	55	"	2.24
3	160	60	Large	6.71
4	161	59	"	6.40
5	158	65	"	11.05
6	157	54	?	

if k=3 so 6 will classify in Large

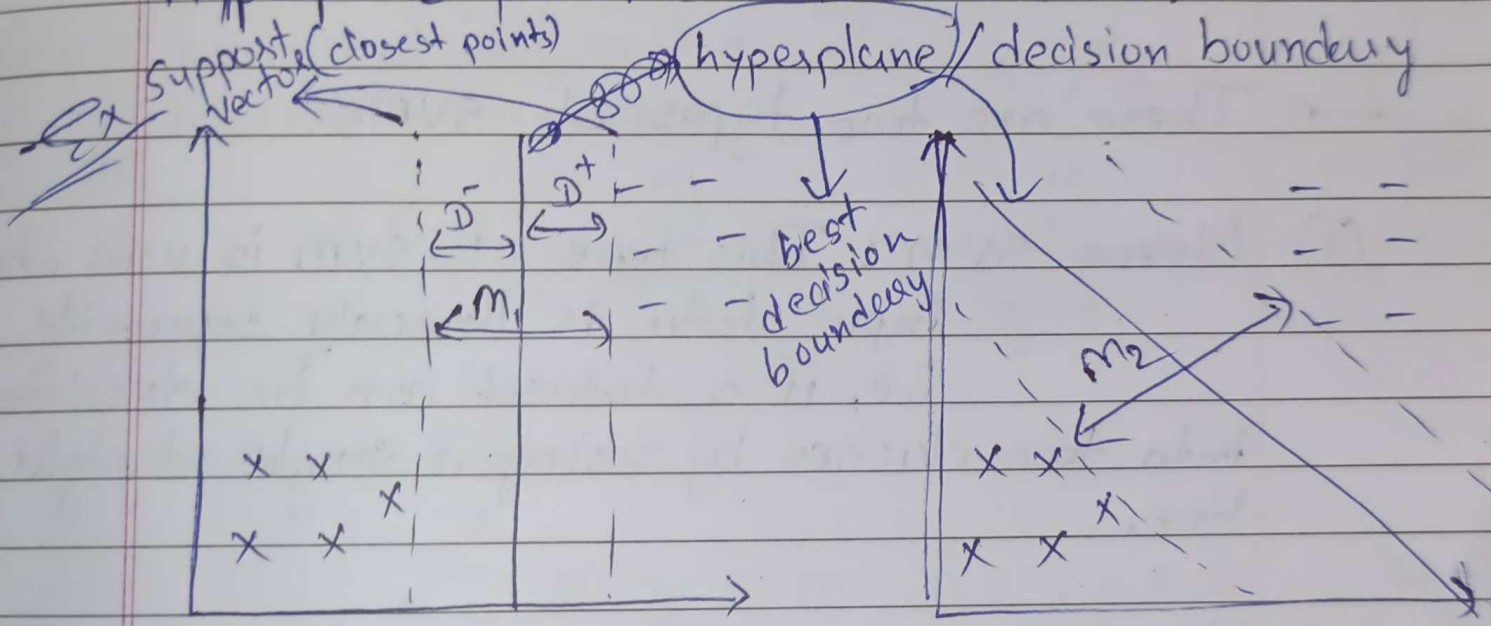
$$d = \sqrt{(157-150)^2 + (54-50)^2}$$

$$= \sqrt{(7)^2 + (4)^2} \quad = 8.06$$

Supervised	Unsupervised
- Linear regression	
- SVM	
- Naive bayes	
- Logistic regression	

★ Support Vector Machine :- (Supervised learning)

⟹ To select which decision boundary we have to consider for SVM **Maximal Margin Hyperplane** parameter is considered.



here; $m_1 << m_2$ So second graph hyper plane is going to be selected for SVM algo

$$\text{Margin} = D^- + D^+$$

Note:- SVM is a classification algorithm and it's job is to perform classification by finding a line/plane that classify the data points in a best way.

✱ SVM Classifier Algorithm

⟹ Supervised learning algorithm
⟹ used to solve both classification & Regression tasks.

⟹ There are two types of SVMs:

(i) Linear SVM: This type of SVM is used when input data is linearly separable, i.e, if a dataset can be classified into two classes by using a single straight line.

(ii) Non-linear SVM: data is not linearly separable

Ex

Given a two-class linearly separable dataset of N points of the form:

$(\vec{x}_1, y_1), (\vec{x}_2, y_2) \cdots (\vec{x}_N, y_N)$

where; y_i's are either +1 or -1

x_1	x_2	class
2	2	-1
4	5	+1
7	4	+1

step1:- Find out 'α' vector $\vec{\alpha} = (\alpha_1, \alpha_2, \cdots, \alpha_N)$ which maximizes

No. of training data

$$\phi(\vec{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1, j=1}^{N} \alpha_i \alpha_j y_i y_j (\vec{x_i} \cdot \vec{x_j})$$

dual objective function

pre-conditions:

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \quad \text{for } i = 1, 2, \cdots, N$$

$$\alpha_1 > 0, \alpha_2 > 0 \cdots \alpha_N > 0$$

So

$$-\alpha_1 + \alpha_2 + \alpha_3 = 0$$

step2: Find weight vector

$$\vec{w} = \sum_{i=1}^{N} \alpha_i y_i \vec{x_i}$$

step 3:- Compute bais

$$b = \frac{1}{2}\left(\min_{i:\, y_i = +1}(\vec{w}\cdot\vec{x}_i) + \max_{i:\, y_i = -1}(\vec{w}\cdot\vec{x}_i)\right)$$

$\min_{i:\, y_i = +1}(\vec{w}\cdot\vec{x}_i)$ → only positive example

$\max_{i:\, y_i = -1}(\vec{w}\cdot\vec{x}_i)$ → only negative example

step 4:- The SVM classifies function is given by

$$f(\vec{x}) = \vec{w}\cdot\vec{x} - b$$

→ eqn of hyperplane

Ex Data:-

x_1	x_2	class
2	1	+1
4	3	-1

here; $N = 2$

$\vec{x}_1 = (2, 1)$

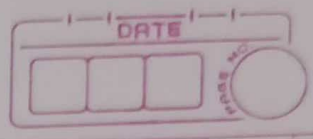
$\vec{x}_2 = (4, 3)$

$y_1 = +1$

$y_2 = -1$

$$f(\vec{x}) = \vec{w}\cdot\vec{x} - b$$

$$\vec{x}_1 \cdot \vec{x}_1 = (2,1) \cdot (2,1)$$
$$= 2 \times 2 + 1 \times 1$$
$$= 5$$



step 1: find $\vec{\alpha} = (\alpha_1, \alpha_2)$

here: $\sum_{i=1}^{N} \alpha_i y_i = 0$

so:

$$\alpha_1 - \alpha_2 = 0$$

$$\therefore \alpha_1 = \alpha_2$$

and:

$$\alpha_1 > 0 \ \& \ \alpha_2 > 0$$

$$\phi(\vec{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1, j=1}^{N} \alpha_i \alpha_j y_i y_j (\vec{x}_i \cdot \vec{x}_j)$$

$$= (\alpha_1 + \alpha_2) - \frac{1}{2} \Big[\alpha_1 \alpha_1 y_1 y_1 (\vec{x}_1 \cdot \vec{x}_1) + \alpha_1 \alpha_2 y_1 y_2 (\vec{x}_1 \cdot \vec{x}_2)$$
$$+ \alpha_2 \alpha_1 y_2 y_1 (\vec{x}_2 \cdot \vec{x}_1)$$
$$+ \alpha_2 \alpha_2 y_2 y_2 (\vec{x}_2 \cdot \vec{x}_2) \Big]$$

$$= (\alpha_1 + \alpha_2) + \frac{1}{2} \Big[\alpha_1^2 (1)(1)(2 \times 2 + 1 \times 1) + \alpha_1 \alpha_2 (1)(-1)(2 \times 4 + 1 \times 3)$$
$$+ \alpha_2 \alpha_1 (-1)(+1)(4 \times 2 + 3 \times 1)$$
$$+ \alpha_2^2 (-1)(-1)(4 \times 4 + 3 \times 3) \Big]$$

$$= (\alpha_1 + \alpha_2) - \frac{1}{2}\left[5\alpha_1^2 - 22\alpha_1\alpha_2 + 25\alpha_2^2\right]$$

⟹ Find values of α_1 and α_2 which maximizes

$$\phi(\vec{\alpha}) = (\alpha_1 + \alpha_2) - \frac{1}{2}\left[5\alpha_1^2 - 22\alpha_1\alpha_2 + 25\alpha_2^2\right]$$

Here; $\alpha_1 = \alpha_2$ so put α_1 instead of α_2.

$$\phi(\vec{\alpha}) = (\alpha_1 + \alpha_1) - \frac{1}{2}\left[5\alpha_1^2 - 22\alpha_1^2 + 25\alpha_2^2\right]$$

$$= 2\alpha_1 - \frac{1}{2}\left[8\alpha_1^2\right]$$

$$= 2\alpha_1 - 4\alpha_1^2$$

Note:- If you want to find out maximization of $\phi(\vec{\alpha})$ you have to considered differentiation.

$$\frac{d\phi}{d\alpha_1} = 2 - 8\alpha_1 = 0$$

So, $\boxed{\alpha_1 = \frac{1}{4} = \alpha_2}$

step2: $\vec{w} = \sum_{i=1}^{N} \alpha_i y_i \vec{x}_i$

$= \alpha_1 y_1 \vec{x}_1 + \alpha_2 y_2 \vec{x}_2$

$= \frac{1}{4}(+1)(2,1) + \frac{1}{4}(-1)(4,3)$

$= \frac{1}{4}(2,1) + \frac{1}{4}(-4,-3)$

$= \frac{1}{4}[(2,1) + (-4,-3)]$

$= \frac{1}{4}(-2,-2)$

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

step3: $b = \frac{1}{2}\left(\min_{i:y_i=+1}(\vec{w}.\vec{x}_i) + \max_{i:y_i=-1}(\vec{w}.\vec{x}_i)\right)$

$= \frac{1}{2}\left((\vec{w}.\vec{x}_1) + (\vec{w}.\vec{x}_2)\right)$

$= \frac{1}{2}\left[\left(-\frac{1}{2}\times 2 - \frac{1}{2}\times 1\right) + \left(-\frac{1}{2}\times 4 - \frac{1}{2}\times 3\right)\right]$

$= -\frac{5}{2}$

step 4:- The SVM classifier function is given by

$$f(\vec{x}) = \vec{\omega} \cdot \vec{x} - b$$

$$= \left(-\frac{1}{2}, -\frac{1}{2}\right) \cdot (x_1, x_2) - \left(-\frac{5}{2}\right)$$

$$= -\frac{1}{2} x_1 - \frac{1}{2} x_2 + \frac{5}{2}$$

$$\boxed{= -\frac{1}{2}(x_1 + x_2 - 5)}$$

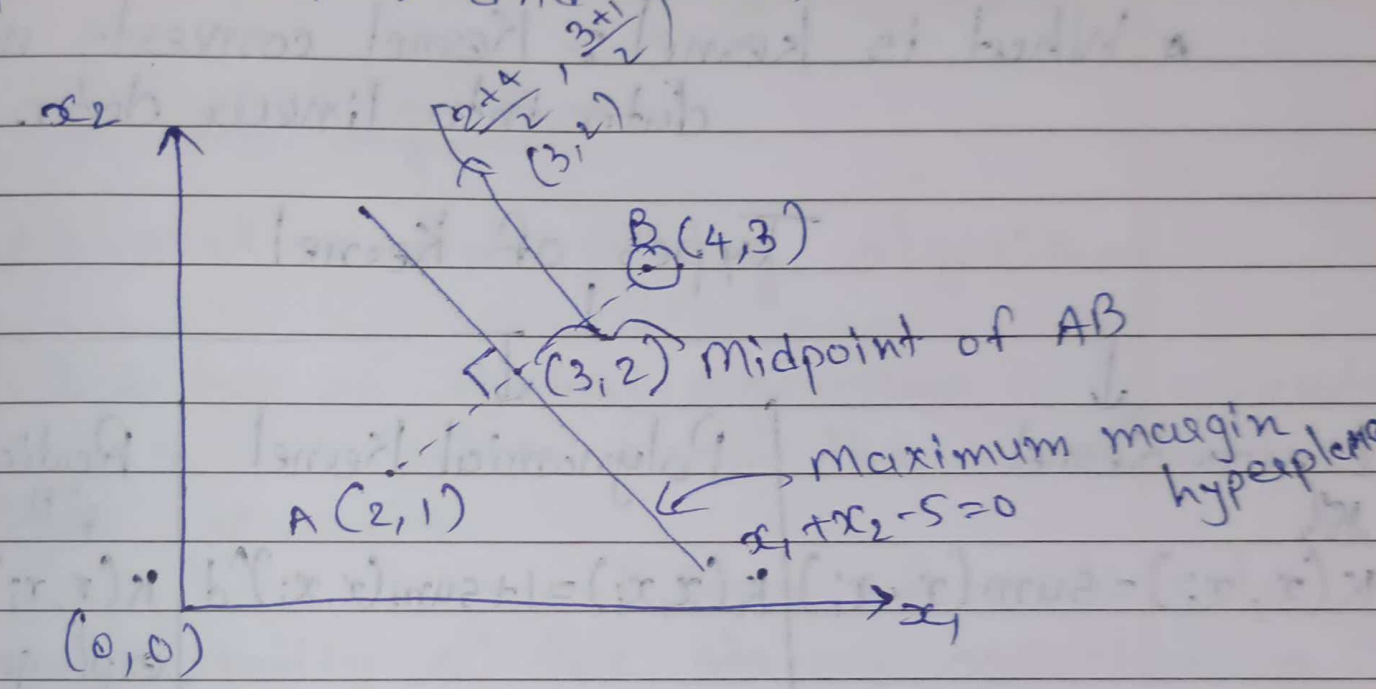
⟹ The equation of the maximal margin hyperplane is

$$f(\vec{x}) = 0$$

$$\therefore -\frac{1}{2}(x_1 + x_2 - 5) = 0$$

$$\therefore x_1 + x_2 - 5 = 0$$

→ The eqn $x_1 + x_2 - 5 = 0$ is the perpendicular bisector of the line segment joining the points (2, 1) and (4, 3)



Ex-2 Linear Example:

x_1	x_2	y
3	1	+1
3	-1	+1
6	1	+1
6	-1	+1
1	0	-1
0	1	-1
0	-1	-1
-1	0	-1

Note:-

⟹ SVM is known for its kernel trick to handle nonlinear data.

* What is kernel :- Kernel converts non-linear data into linear data.

Types of Kernel

Linear Kernel	Polynomial Kernel	Radial Basis fn kernel
$K(x, x_i) = sum(x \cdot x_i)$	$K(x, x_i) = 1 + sum(x \cdot x_i)^d$	$K(x, x_i) =$ exp(-gamma * sum((x - x_i^2))

(4) Sigmoid Kernel

$f(x_1, x_2) = \tanh(\alpha x^T y + x)$

Imp note:-

high bias ⟺ low bias :- Before we allowed misclassification we picked a threshold that was very sensitive to the training data.

low variance ⟺ high variance :- It performed poorly when we got new data.

* When we allow misclassifications, the distance between the observations & the threshold is called a Soft Margin.

☆ Apriori Algorithm :-

⟹ Association Rule Mining algorithm.

⟹ Objective of Apriori algorithm is to generate association rules from the frequent itemsets.

⟹ Calculate the confidence of each rule and identify all the strong association rules.

⟹ The association rule describes how two or more objects are related to one another.

* Support:
- Support refers to the default popularity of any product.

Ex: Support(item) = Trans. of item / Total trans.

* Confidence
- Confidence refers to the possibility that the customers bought multiple items together.

Ex

Transaction ID	Items Bought
1	{Bread, Butter, Milk}
2	{Bread, Butter}
3	{Beer, Cookies, Diapers}
4	{Milk, Diapers, Bread, Butter}
5	{Beer, Diapers}

Min_support = 40%.
Min_confidence = 70%.

Soln:-

step-1 Unique Itemsets are: Bread, Butter, Milk, Diapers, Beer

1-Itemset	Support_count
Bread	3
Butter	3
Milk	2
Beer	2
Cookies	1
Diapers	3

→ how many time the item is bought from given transaction.

Min_support_count = Min_support × (no. of transactions / itemset count)
= 40% × 5
= $\frac{40 \times 5}{100}$ = 2

→ Min-Support_Count is 2 that indicates that 1-Itemset must have support count greater than or equal to 2 then and then it is called 1-frequent Itemset.

→ These cookies has support_count value 1 so we eliminate that itemset

1-Frequent Itemset	Support_count
Bread	3
Butter	3
Milk	2
Beer	2
Diapers	3

step-2:- Create combinations for 2-Itemset from ~~1-frequent Itemsets~~. unique itemset of above table.

Ex

{Bread, Butter}, {Bread, Milk}, {Bread, Beer}
{Bread, Diapers}, {Butter, Milk}
{Butter, Beer}, {Butter, Diapers}, {Milk, Beer}
{Milk, Diapers}, {Beer, Diapers}

2-Itemset	Support_count	
{Bread, Butter}	3	
{Bread, Milk}	2	
{Bread, Beer}	0	X
{Bread, Diapers}	1	X
{Butter, Milk}	2	
{Butter, Beer}	0	X
{Butter, Diapers}	1	X
{Milk, Beer}	0	X
{Milk, Diapers}	1	X
{Beer, Diapers}	2	

2-Frequent Itemset	Support_count
{Bread, Butter}	3
{Bread, Milk}	2
{Butter, Milk}	2
{Beer, Diapers}	2

step:-3 Unique items : Bread, Butter, Milk, Beer, Diaper

3-Itemset	Support-count	
Bread, Butter, Milk	2	
Bread, Butter, Diapers	1	x
Bread, Butter, Beer	0	x
Bread, Milk, Diapers	1	x
Bread, Milk, Beer	0	x
Bread, Diapers, Beer	0	x
Butter, Milk, Diapers	1	x
Butter, Milk, Beer	0	x
Butter, Diapers, Beer	0	x
Milk, Diapers, Beer	0	x

3-Frequent Itemset	Support Count
Bread, Butter, Milk	2

→ From above 3-frequent Itemset table we can say that it is not possible to generate 4-Itemset because unique itemset available is only 3. So it will stop here and now it will generate strong association rule.

step:-4 To find strong association rule.

min. confidence = 70%.

Confidence $(X \rightarrow Y) = P(Y|X) = \frac{P(X \cup Y)}{P(X)}$

↳ It means this much percent of customers who bought X bought Y also.

⟹ Here, we have 5 frequent itemsets i.e.

{Bread, Butter}, {Bread, Milk}, {Butter, Milk}, {Diapers, Beer}. and {Bread, Butter, Milk}

Therefore, candidate rules are:

- For {Bread, Butter}

$P(\text{Bread} | \text{Butter}) = \frac{3}{3} = 100\%$ (Strong)

$P(\text{Butter} | \text{Bread}) = \frac{3}{3} = 100\%$ (Strong)

- For {Bread, Milk}

$P(\text{Milk} | \text{Bread}) = \frac{2}{3} = 67\%$

$P(\text{Bread} \mid \text{Milk}) = \frac{2}{2} = 100\%$ (Strong)

- For {Butter, Milk}

$P(\text{Butter} \mid \text{Milk}) = 2/2 = 100\%$ (Strong)

$P(\text{Milk} \mid \text{Butter}) = 2/3 = 67\%$

⟹ Same way do for {Diaper, Beer} and {Bread, Butter, Milk}.

$P(\text{Bread, Butter} \mid \text{Milk}) = \frac{2}{2} = 100\%$ (Strong)

$P(\text{Milk, Butter} \mid \text{Bread}) = \frac{2}{3} = 67\%$

$P(\text{Milk, Bread} \mid \text{Butter}) = \frac{2}{3} = 67\%$

$P(\text{Milk} \mid \text{Bread, Butter}) = \frac{2}{3} = 67\%$

$P(\text{Bread} \mid \text{Milk, Butter}) = \frac{2}{2} = 100\%$ (Strong)

$P(\text{Butter} \mid \text{Milk, Bread}) = 2/2 = 100\%$ (Strong)

☆ Locally Weighted Regression Algorithm:

⟹ Locally weighted regression is instance based learning algorithm.

Local:- Because it will not considered all the data points, it will only consider near data points with the query point.

Weighted:- Because the contribution of each training example is weighted by its distance from the query point.

Regression:- Because this is the term used widely in the statistical learning community for the problem of approximating real-value functions.

⟹ Consider, locally weighted regression in which the target function f is approximated near x_q using a linear function of the form ...

query point ←

$$\hat{f}(x) = w_0 + w_1 a_1(x) + \cdots + w_n a_n(x)$$

where;

$a_i(x)$ = value of i^{th} attribute of the instance x.

⟹ In locally weighted regression initially weights are assigned randomly and to fine tuned this weights Gradient Descent algorithm of ANN is used.

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

Error fⁿ in ANN; t_d ↓ target o/p; o_d ↳ predicted o/p.

$$E = \frac{1}{2} \sum_{x \in D} [f(x) - \hat{f}(x)]^2$$

E ↙ Error fⁿ in locally weighted regression

$$\Delta w_j = \eta \sum_{x \in D} [f(x) - \hat{f}(x)] a_j(x)$$

$$w_j = w_j + \Delta w_j$$

↗ weight updation rule

Locally Weighted Regression algorithms

Approach-1:- Locally

Steps:- Minimize the squared error over just the k-nearest neighbors.

$$E_1(x_q) = \frac{1}{2} \sum_{x \in k \text{ nearest of } x_q} (f(x) - \hat{f}(x))^2$$

Approach-2:- weighted

Minimize the squared error over the entire set D of training examples, while weighting the error of each training example by some **decreasing function** K of its distance from x_q:

$$E_2(x_q) = \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2 K(d(x_q, x))$$

Approach-3:- Locally weighted (Combine 1 & 2)

$$E_3(x_q) = \frac{1}{2} \sum_{x \in k \text{ nearest neighbour of } x_q} (f(x) - \hat{f}(x))^2 K(d(x_q, x))$$

$$\Delta w_j = \eta \sum_{x \in k \text{ nearest neighbors of } x_q} K(d(x_q, x))(f(x) - \hat{f}(x))\, a_j(x$$

$$w_j = w_j + \Delta w_j$$