28-02-'19

UNIT-IV: LEARNING FROM EXAMPLES, LEARNING PROBABILISTIC MODELS, LEARNING WITH HIDDEN VARIABLES

Learning From Examples

Forms of learning

- Any agent can improve its learning by observing the data.
- We have to consider four factors for an agent to learn, i.e., to improve the agent:
  (i) Which component is to be improved in the agent?
  (ii) What prior knowledge does the agent already have?
  (iii) What representation is used for data and components in the agent?
  (iv) What feedback is available to learn from for the agent?
- The agent has to learn six components.

C1 (i) A direct mapping from conditions or current state to actions.

C2 (ii) A means to infer relevant properties of the world from percept sequence.

C3 (iii) Information about the way the world evolves and about the results of possible actions the agent can take.

C4 (iv) Utility information indicating desirability of world states.

C5 (v) Action value information indicating the desirability of actions.

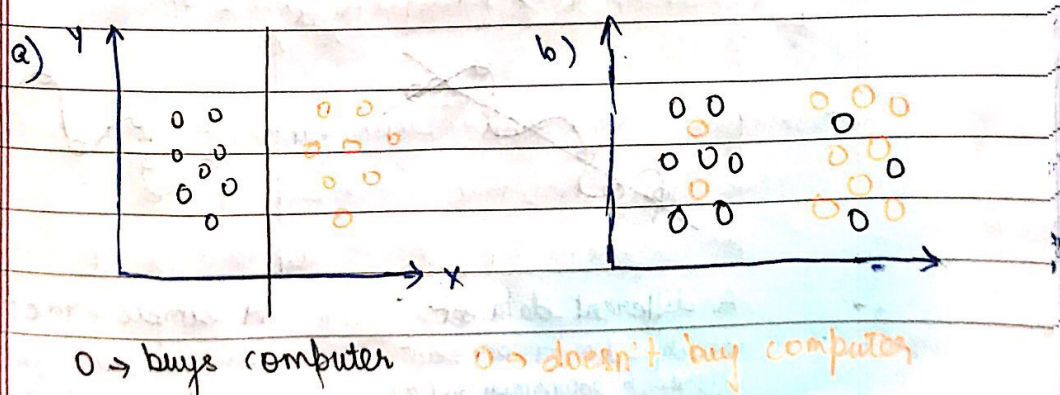
C6 (vi) Goals that describe classes of states whose achievement maximizes the agent's utility

- Representation and Prior Knowledge

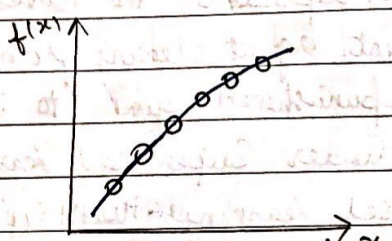
(i) Factored Representation

- We have a vector of attribute values which has both input values and output values.
- The values may be discrete or continuous.
- Analytical / Deductive Learning: From known general rule to new rule.
- Inductive Learning: Learning general rules with specific ~~and~~ input and output we get next state.
- Unsupervised Learning - Learn without any feedback
- Semi Supervised Learning - Learn from labels and label unlabelled instances.
- Reenforcement Learning - Learn from the environment, depends on the moves taken by the agent, agent learns from rewards and /or punishments given to it.
- C1 comes under Supervised Learning
- ~~Semi~~ Supervised Learning - With i/p, o/p pairs, we must match input to output.

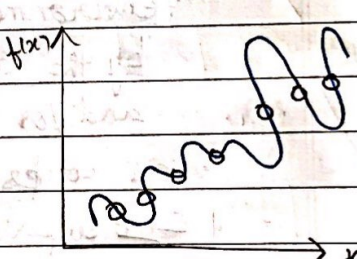
- Supervised Learning



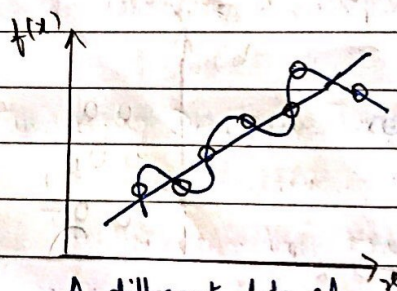
- (a) requires a polynomial function with one degree → simple.
- (b) requires a polynomial function with six or seven degrees → complex.
- Supervised learning:
  N example i/p - o/p pairs
  $(x_1, y_1), (x_2, y_2), \ldots\ldots, (x_N, y_N)$
  $y = f(x)$
  h - hypothesis
- Learning - Agent searches from the search space such that it performs well.
- We have to consider a simple hypothesis so that the result is accurate.
- Types of hypothesis



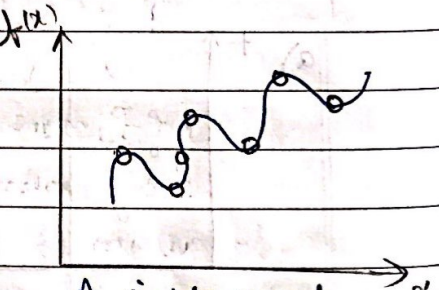
x, f(x) pairs and Consistent Linear Hypothesis



Consistent, degree-7 polynomial hypothesis for sample data set



A different data set which admits an exact degree 6 polynomial fit or approximate linear fit



A simple exact sinasoidal fit to same data set

- We have to check how probable the hypothesis is to that data.

$$h^* = \underset{(h \in H)}{\operatorname{argmax}} P(h/\text{data})$$

By Baye's Rule

$$h^* = \underset{(h \in H)}{\operatorname{argmax}} P(\text{data}/h)\, P(h)$$

→ When the degree is low, the prior probability is high.

→ For a high degree polynomial function, the prior probability is low (less priority).

- A simple hypothesis phase can be terminated easily.
- Classification is categorization, but Regression is prediction.

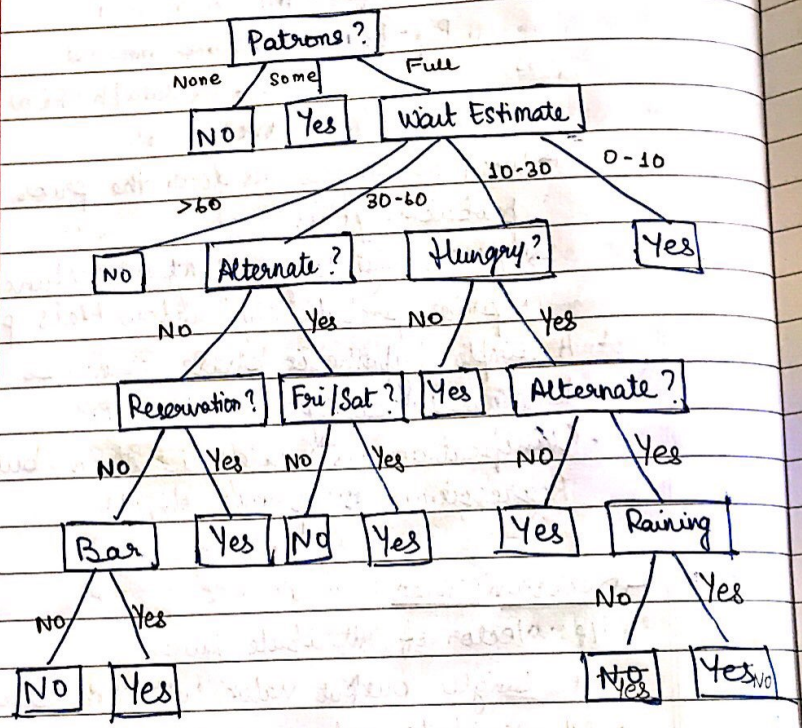
→ **– Decision Tree**

- i/p: Vector of Attribute Values
- o/p: Single output value with one decision
- All attribute values are given as input, it classifies and gives a decision as an output.
- Boolean Classification is done here.
- Eg: Build a decision tree to decide whether to wait for a table at a restaurant

→ Attributes: Alternate (other restaurants nearby), Bar (available or not), Friday / Saturday, Hungry, Patrons (ppl in restaurant), Price (of restaurant), Raining (outside), Reservation (done or not), Type (French, Italian, etc.), Wait Time (00-10, 10-20, 20-30, 30-40 min)

→ Ignore 'price' and 'type' attribute while constructing the decision tree.

→ Decision Tree (Boolean):



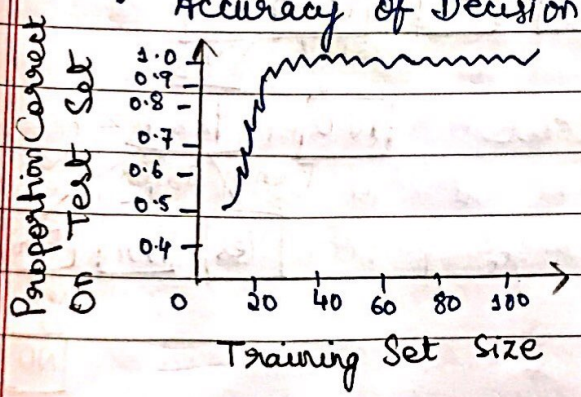
01-03-'19

- Expressiveness of Decision Trees

  Goal $\Leftrightarrow$ (Path 1 V Path 2 V .....)

- Inducing Decision Trees from Examples
  - We have positive examples and negative examples.
  - The decision tree must be as simple as possible and we must get a path.

- So, we go for a decision tree learning algorithm such that it finds a decision tree which is as simple as possible, so that the depth is minimum.
- It considers the attributes that are of first priority. It eliminates/does not consider useless attributes, and ignores them so that we can construct a simple decision tree.
- If all the examples that we have are positive (eg. an unfair coin), we get a decision and we don't need to construct a tree.
- If all the examples are empty, then return plurality value; i.e., most common output values among set of values (tie-breaker).
- If all examples have same classification, then return that type of classification.
- If the examples are half positive and half negative, we must consider the best attribute that can be considered.
- If there are no attributes in our decision tree, it means that there is some error or noise in the data, so we cannot construct the decision tree.
- Accuracy of Decision Tree Learning Algorithm



* Accuracy is high since each and everything is done 20 times.

-Decision Tree Learning Algorithm

function Decision_Tree_Learning (examples, attributes, parent_examples) returns a tree

if examples is empty return PLURALITY_VALUE (parent_examples)

else if all examples have same classification then return the classification

else

A ← aggmax IMPORTANCE (a, examples)
a ∈ attributes

tree ← a new decision tree with root test A

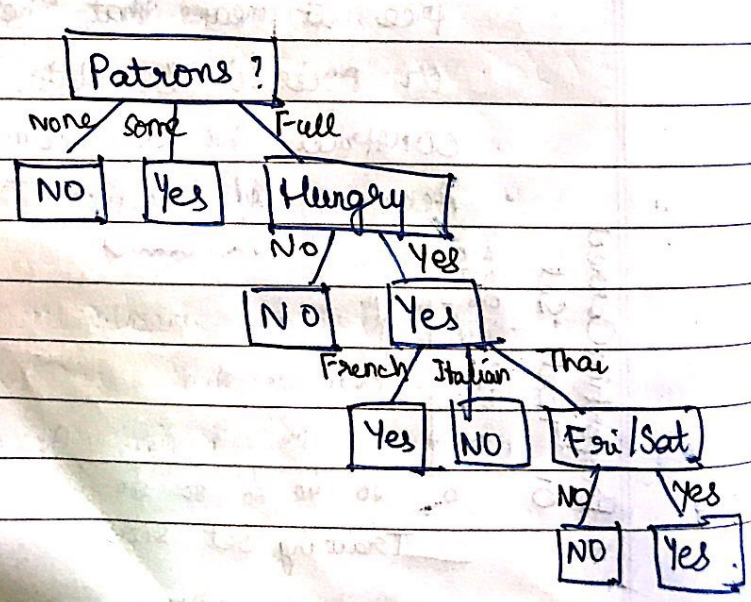
for each value v_k of A do

exs ← {e : e ∈ examples and e.A = v_k}

Subtree ← Decision_Tree_Learning (exs, attributes − A, examples)

add a branch to tree with label (A = v_k) and subtree

return tree



- Choosing Attribute Tests
  - We must consider only important attributes in the decision tree such that it becomes simple.
  - We do this using entropy and calculating information gain.
  - Entropy is randomness.
  - For a fair coin, entropy = 2.
  - For an unfair coin, entropy = 0.
  - Entropy is the uncertainity of a random variable.
  - Entropy : H(V)

$$H(V) = \sum_k P(V_k) \log_2 \frac{1}{P(V_k)} = -\sum_k P(V_k) \log P(V_k)$$

$$H(\text{Fair}) = -(0.5 \log_2 0.5 + 0.5 \log_2 0.5) = 1$$

$$H(\text{loaded}) = -(0.99 \log_2 0.99 + 0.01 \log_2 0.01) \approx 0.08 \text{ bits} \approx 0.08 \text{ bits}$$

(unfair coin = 99% head)

  - Entropy of a Boolean Random Variable

$$B(q) = -[q \log_2 q + (1-q) \log_2 (1-q)]$$

Goal Attributes

$$H(\text{Goal}) = B\left(\frac{p}{p+n}\right)$$

→ For 'k' distinct values

$$B\left(\frac{p_k}{p_k + n_k}\right)$$

To know imp attributes to be chosen for decision tree and remove unnecessary ones

  - Randomly chosen example from training set · $\frac{(p_k + n_k)}{(p+n)}$
  - Gain(A) = $B\left(\frac{p}{p+n}\right)$ − Remainder (A) [Information Gain]

where Remainder(A) = $\sum_{k=1}^{d} \frac{p_k + n_k}{p+n} B\left(\frac{p_k}{p_k + n_k}\right)$

- Generalization and Overfitting
  - When there is no target function, we get all the attributes in the decision tree and it will be overfit.
  - For an overfit decision tree, we have to cut off unnecessary branches called as pruning.
  - We check from the leaf and prune all unnecessary test nodes, until we find a decision tree that is relevant and simple.
  - How large should Information Gain be?
    - → Start with NULL hypothesis.
    - → If the degree of deviation is less than 5% of probability, then NULL hypothesis is accepted / accurate.
    - → Else, it is rejected.
  - The degree of deviation can be calculated as:

$$\hat{p}_k = p \times \frac{p_k + n_k}{p+n} \qquad \hat{n}_k = n \times \frac{p_k + n_k}{p+n}$$

[p = +ve, n = -ve]

Total Deviation:

$$\Delta = \sum_{k=1}^{d} \frac{(p_k - \hat{p}_k)^2}{\hat{p}_k} + \frac{(n_k - \hat{n}_k)^2}{\hat{n}_k}$$

  - Early Stopping: Stop generating that particular node if it is not required.
  - If an attribute is not good enough, we should not split it again and early stop generating that particular node.

- Broadening the Applicability of Decision Trees

- Issues in Decision Trees:
  (i) Missing Data
  (ii) Multivalued Attributes
  (iii) Continuous and integer valued input attributes
  (iv) Continuous valued output attributes

- Missing Data
  → Examples may be unrecorded/expensive to record.
  → How to classify examples?
  → How to calculate information gain for missing data.

- Multivalued Attributes
  → Many possible values for an attribute.
  → Inappropriate values for information gain.
  → Check each and every value, gain ratio.

- Continuous and Integer valued input attributes
  → Split the attributes for continuous attributes.
  → But it is costly.

- Continuous valued output attributes
  → When we want to get an integer output such as predicting the value of a flat, we go for regression rather than classification.

Date
Page

- Evaluating and Choosing the Best Hypothesis

- It connects from the past to the future.
- It may take the past evidences, but a hypothesis must be selected in such a way that the future is best fit.
- The examples that satisfy these assumptions are called independent identically distributed (i.i.d), which connect the past to the future.
- $P(E_j | E_{j-1}, E_{j-2}, \dots) = P(E_j)$
  $P(E_j) = P(E_{j-1}) = P(E_{j-2}) = \dots$
- If we give already seen evidences in test set, then the accuracy is less. We must give unseen evidences. => Best Fit
- Hold-out cross validation: We give the test set from the training set, 100% accuracy
- K-fold cross validation: Divide into 'k' subsets, at each ith iteration give i 1/k as test set, do k rounds such that the accuracy is more.
  Popular values for k = 5, 10
- Leave One Out Cross Validation to get a more accurate point.
- Peeking: Using test set performance to both choose a hypothesis and evaluate it. We wait till the time the lock is released to validate the data.

- Model Selection Complexity vs Goodness of Fit
  - → We cannot estimate the accuracy of the model based on the degree of the polynomial.
  - → Model Selection and Optimization is finding the best hypothesis within the given space.
  - → We have a wrapper. It takes a decision tree learning algorithm as a parameter. The wrapper enumerates models depending on the parameter and size. It uses cross validation and it checks the average error rate on training and test set.
  - → The training set error decreases when the validation decreases.
  - → The model can be fit when the error rate is less.
  - → We decrease the size of the dataset.

- From error rates to loss
  - → Generalization loss for Hypothesis

$$GenLoss_L(h) = \sum_{(x,y)\in\mathbb{E}} L(y, h(x))\, P(x,y)$$

  - → Minimum Expected Generalization loss

$$h^* = \underset{h\in H}{\operatorname{argmin}}\ GenLoss_L(h)$$

  - → Empirical Loss

$$EmpLoss_{L,E}(h) = \frac{1}{N} \sum_{(x,y)\in E} L(y, h(x))$$

$$\hat{h}^* = \underset{h\in H}{\operatorname{argmin}}\ EmpLoss_{L,E}(h)$$

• Regularization

$$Cost(L) = Emploss(h) + \lambda\, Complexity(h)$$

$$\hat{h}^* = \underset{h \in H}{\operatorname{argmin}}\ Cost(h)$$

07-03-'19

→ **Theory Of Learning - Learning with an agent**

- How many examples do we need to get a good hypothesis?
- What hypothesis should we use?
- How complex should the hypothesis space be?
- How to avoid overfitting?
- How may examples do we need for learning?
- If the hypothesis is wrong, we get wrong result
- If our hypothesis is consistent with a large number of examples, then it is right.
- The Probably Accurate Approximately Correct (PAC) learning algorithm: If our hypothesis is consistent with a large number of axioms, then the error rate is ~~to~~ less.
- Error Rate

$$Error(h) = GenLoss_{L_{0/1}}(h) = \sum_{x,y} L_{0/1}(y, h(x))\, P(x,y)$$

error $(h) \leq \epsilon$ => Hypothesis is Approximately Correct

ϵ => small constant

error $(h_b) > \epsilon$ => Hypothesis not approximately correct

- $P(h_b$ agrees with N example$) \leq (1-\epsilon)^N$

$P(H_{bad}$ contains a consistent hypothesis$) \leq |H_{bad}|(1-\epsilon)^N$

$\leq |H|(1-\epsilon)^N$

Reduce Probability

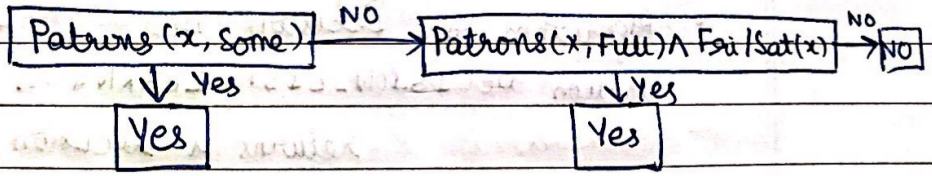
$|H|(1-\epsilon)^N \leq \delta$

$1-\epsilon \leq e^{-\epsilon}$

$N \geq \frac{1}{\epsilon}\left(\ln\frac{1}{\delta} + \ln|H|\right)$ → Algorithm is consistent, Hypothesis is correct

N: No of examples H: Hypothesis ϵ: Small consistent	Equation should see more than 'n' examples for our algorithm to be consistent

- PAC learning examples: learning decision lists
  - Applying PAC algorithm for new hypothesis space.
  - Decision ~~tree~~ list is like a decision tree but less complex. It branches only in one direction.
  - It is a series of tests. If it passes the tests, it gives the result, i.e., one branch.
  - Decision List for Restaurant Problem:



Will Wait $\Leftrightarrow$ (Patrons = Some) $\vee$ (Patrons = Full $\wedge$ Fri/Sat)

- We have a k-dimensional decision test.
- Our decision list above is 2-dimensional, since it has only 2 literals.
- A k-dimensional decision list or KDL has at most k literals and then it can generalize successfully.

08-03-'24

- We have two main tasks in k-DL:
  (i) We have to show that KDL is learnable.
  (ii) We have to find an efficient algorithm that returns a consistent decision list.

- Number of conjunctions:

$$|Conj(n,k)| = \sum_{i=0}^{kd} \binom{2n}{i} = O(n^k)$$

$$|k\text{-}DL(n)| = 2^{O(n^k \log_2(n^k))}$$

$$n \geq \frac{1}{\epsilon}\left(\ln\frac{1}{\delta} + O(n^k \log_2(n^k))\right)$$

↳ we need 'n' no. of examples atleast for the PAC algorithm to learn from the examples and become accurate.

- To find an efficient algorithm, we use greedy algorithm such that it is accurate. It considers only the examples that are relevant and removes or does not consider the other examples.

- Algorithm for Decision Tree:

function DECISION_LIST_LEARNING (examples)
    returns a decision, or false
if examples is empty then return the decision list NO
t ← a test that matches a non empty subset examples_t of examples
such that the members of examples_t are all positive or all negative
if there is no such t then return false
if the examples in examples_t are positive then else O ← YES O ← NO
return a decision list with initial test t and outcome O
and remaining test given by DECISION_LIST_LEARNING (examples − examples_t)

we try to
the lo
each de

A = de
Ro

05-03-'19

→ Regression and Classification with Linear Models

- We use regression to predict the future with the help of the past.
- Univariate Linear Regression:
  - i/p: x o/p: y

$y = w_1 x + w_0$ Here, w_1 and w_0 = weights

Vector : $w = (w_0, w_1)$ $h_w(x) = w_1 x + w_0$

h → hypothesis

$$Loss(h_w) = \sum_{j=1}^{N} L_2(y_j, h_w(x_j)) = \sum_{j=1}^{N} (y_j - h_w(x_j))^2$$

$$= \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2$$

$w^* = \text{argmin}_w \, Loss(h_w)$

$\sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2$ is minimized w_0 and $w_1 = 0$

$w_1 = 0.232$ $w_0 = 246$ → For given fig

$w_0 = (\Sigma y_j - w_1 (\Sigma x_j))/N$

$$w_1 = \frac{N(\Sigma x_j y_j) - (\Sigma x_j)(\Sigma y_j)}{N(\Sigma x_j^2) - (\Sigma x_j)^2}$$

We have to use gradient descent to minimize loss.

We have to make the weights of w_0 and w_1 near to 0 to minimize the loss

- Gradient Descent Agorithm:

$w \leftarrow$ any point in parameter space

loop until convergence do

for each w_i in w do

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(w)$$

We try to minimize the loss at each descent

α = Learning Rate

classmate
Date
Page

- In univariate, the loss function is a quadratic function: (Partial Derivation for the linear fn., for one training eg. with x,y)

(x, y)

$$\frac{\partial}{\partial w_i} Loss(w) = \frac{\partial}{\partial w_i} (y - hw(x))^2$$

$$= 2(y - hw(x)) \times \frac{\partial}{\partial w_i} (y - hw(x))$$

$$= 2(y - hw(x)) \times \frac{\partial}{\partial w_i} (y - (w_1 x + w_0))$$

$$\frac{\partial}{\partial w_0} Loss(w) = -2(y - hw(x))$$

$$\frac{\partial}{\partial w_1} Loss(w) = -2(y - hw(x)) \times x$$

Unspecifity learning Rate α

$w_0 \leftarrow w_0 + \alpha (y - hw(x))$

$w_1 \leftarrow w_1 + \alpha (y - hw(x)) \times x$

If x = +ve → Reduce w_1

If x = -ve → Increase w_1

- Multivariate Linear Regression

$hw(x_j) = w_0 + w_1 x_{j,1} + \dots + w_n x_{j,n}$

$= w_0 + \sum_i w_i x_{j,i}$

- Augmented Vectors: Add a feature to each x by taking on a: x_j $1 : x_{j,0} = 1$

Then: $h_w(x_j) = w \cdot x_j = w^T x_j = \sum_i w_i x_{j,i}$

- And batch gradient descent update becomes.

$w_i \leftarrow w_i + \alpha \sum_j (y_j - hx(x_j)) x_{j,i}$

classmate
Date
Page

. Regularization - Used to Remove overfitting.

Cost(h) = EmpLoss(h) + λ Complexity(h)

Complexity(h_w) = $L_q(w) = \sum_i |w_i|^q$

It will not intersect on axes.

In L1, weights will become 0, but not in L2.

L1: Intersection done with diamond box - intersects on axes - weights will be zero - minimum loss.

L2: Intersection done with circle - intersection not on axes - weights are not zero - no minimum loss

- Linear Classification - Hard Tresholds
  - Decision Bondary
  - Linearly Separable
  - Classification Hypothesis

- Perceptron Learning Rule
  - Perceptron Convergence Theorem
  - For a single example (x, y)

$$w_i \leftarrow w_i + \alpha (y - h_w(x)) x_j$$

- Linear Classification with Logistic Regression

$$h_w(x) = \text{Logistic}(w, x) = \frac{1}{1 + e^{-w \cdot x}}$$

For a single sample (x, y) and L_2 loss function

$$w_i \leftarrow w_i + \alpha (y - h_w(x)) h_w(x) (1 - h_w(x)) x_i$$

14-03-'19

→ Artificial Neural Networks

- Machine learning approach where a machine works like a human brain.
- It has neurons which are connected to each other.
- Thus, ANN is a network connected with a large no. of neurons, associated with some weights.
- It is called as connectionist model.
- The neuron is the main component.

→ Neuron Model
→ Architecture
→ Learning Algorithms

- Neuron model is the processing unit.
- Architecture has neurons connected to each other with weights.
- Learning algorithm makes the neurons learn.

→ Neuron Model

* Inputs
* Weights
* Adder
* Activation Function
* $U = \sum_{i=1}^{m} w_i * x_i$

op: $Y = \varphi(U+b) = \varphi(V)$

Here:

U = linear combiner - sum of wts x inputs

m = no. of inputs

φ = Activation fn to limit outputs

$V = u + b$

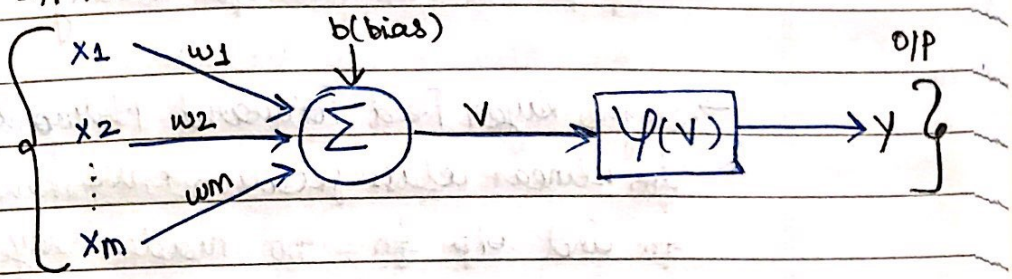
Activation fn: $v = \sum_{i=0}^{m} w_i * x_i$ where $w_0 = b$ and $x_0 = 1$

Here: b = bias

w = weights

x = inputs

→ Neuron



→ Activation Functions

- Step Function

$$\varphi(v) = \begin{cases} a, & \text{if } v < c \\ b, & \text{if } v \geq c \end{cases}$$

- Ramp Function

$$\varphi(v) = \begin{cases} a, & \text{if } v < c \\ b, & \text{if } v > d \\ a + [(v-c) * (b-a)/(d-c)], & \text{otherwise} \end{cases}$$

- Sigmoid Function

$$\varphi(v) = z + \frac{1}{1 + \exp(-x * v + y)}$$

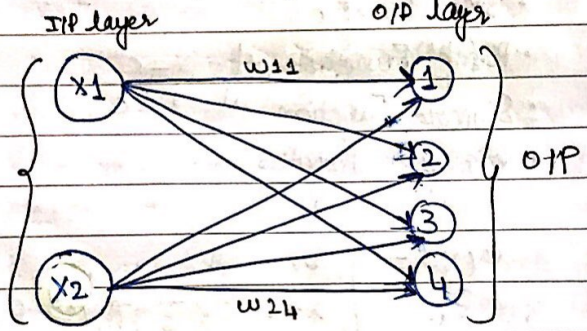
- Gaussian Function

$$\varphi(v) = \exp\left(-\frac{(v-c)^2}{\sigma^2}\right)$$

→ Neural Network Architecture
- Single Layer Feed Forward Network
- Multi Layer Feed Forward Network
- Recurrent Network

→ Step function is used for Binary Classification.

→ Single Layer Feed Forward Network is used for linear classification. We have only I/P fn. and o/p fn., no hidden layers.
- I/P is connected to links, where we have the sum of weights.

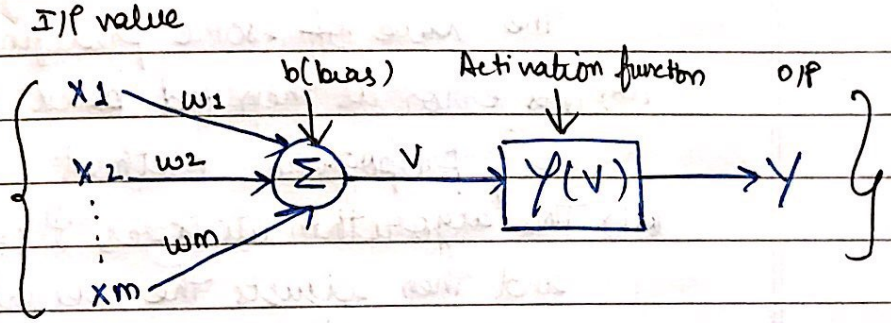


Treshold Value.
O/P Value > Treshold value, neuron takes activated value.
O/P Value < Treshold value, neuron takes deactivated value.

→

→ Perceptron Neuron Model

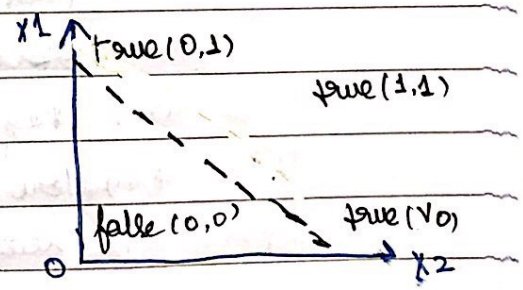
- Special model of single layer feed forward network
- Used for binary classification.
- Classifies input into two categories and gives one O/P.
- Step fn. is activation fn.
- Before activation fn. is used, it must be trained.
- If it classifies unknown instances correctly, output / accuracy is high.
- +ve O/P : C1 -ve O/P : C2



$$\varphi(V) = \begin{cases} -1, & \text{if } V < \theta \\ +1, & \text{if } V \geq \theta \end{cases}$$

- OR function graph → linearly separates T and F

I/P		O/P
X1	X2	X1 OR X2
0	0	0
0	1	1
1	0	1
1	1	1



- Limitations : Can model only linearly separable functions, cannot go to more extent, only two classes used, can model AND by not XOR function.

→ Multi Layer Feed Forward Network

- We have a hidden layer here.
- Used for data which is not linearly separable.
- It will evolve with continuously learning examples.
- We have a backpropogation method
- Learning Technique: Backpropogation Method

(i) The output values obtained for a known set of input values is compared with the correct answer in order to determine the value of some predefined error function

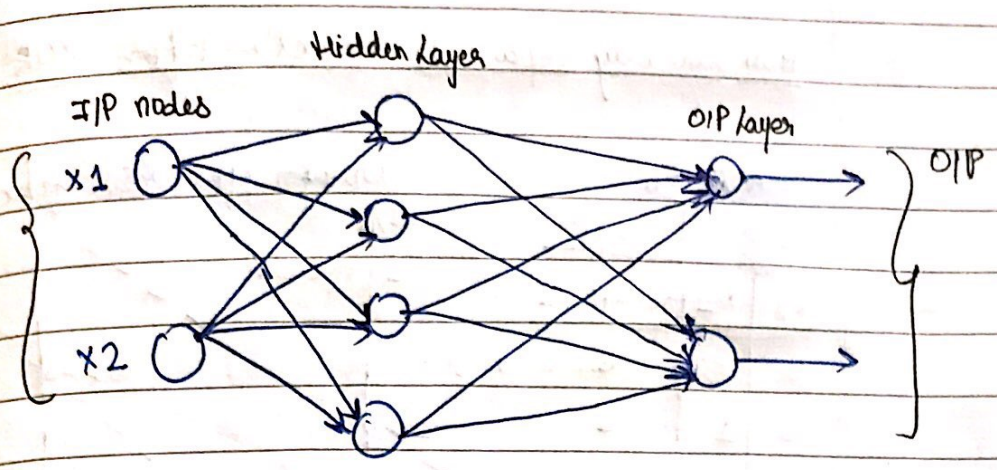
(ii) This error is then fed back to the network using propogation method.

(iii) The algorithm utilizes this information and then adjusts the weights of all connections so that the corresponding error function is reduced to some extent

(iv) This procedure is repeated for a large number of training cycles. So the network obtains a state where the error becomes negligible.

(v) You can say that the network has learnt target function.

P.T.O. →



→ Learning Algorithm for Perceptron

- Initial random weights as +0.5 and −0.5.
- Training data is presented to perceptron and the output is observed.
- We check the output if it is incorrect, the weights are adjusted
  - A = learning rate
  - O/P correct => A = 0

    O/P too low => A = +ve no

    O/P too high => A = −ve no
  - $w_i = w_i + (A * X_i * E)$

    E = Error A (0 < A < 1) → learning rate

    ↳ Adjustment of weights
- Again, training data is checked and if the output is wrong, weights are adjusted again.
- Training continues till all weights are correct and all errors are zero.
- Each iteration in this process is known as an 'epoch'.

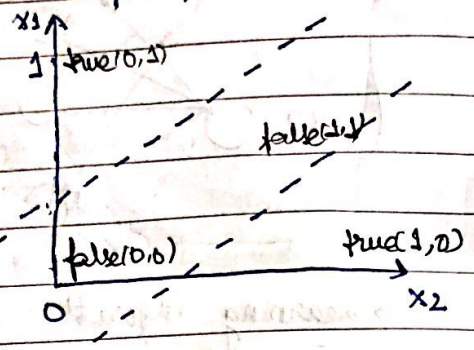
15-03-'19

Non Linearly Separable Function for XOR function

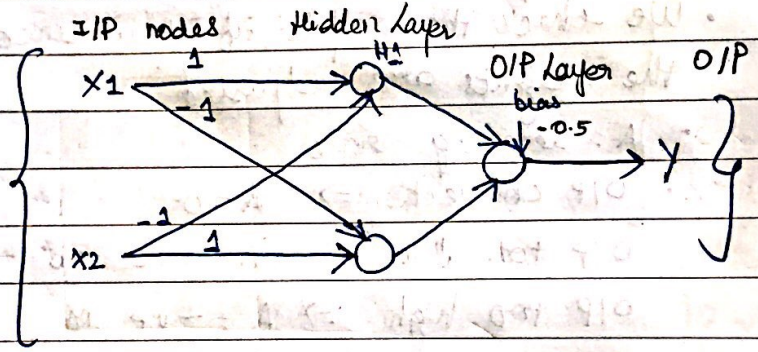
XOR Fn

I/P		O/P
X1	X2	X1 XOR X2
0	0	0
0	1	1
1	0	1
1	1	0

Graph for XOR function



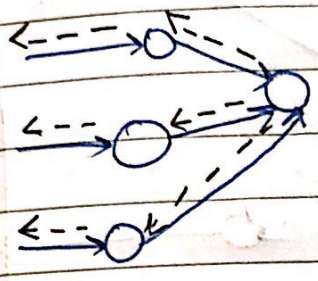
Network for XOR



I/Ps		O/Ps of Hidden Layer		O/P Node	X1 XOR X2
X1	X2	H1	H2		
0	0	1	0	$-0.5 \rightarrow 0$	0
0	1	$-1 \rightarrow 0$	1	$0.5 \rightarrow 1$	1
1	0	1	$-1 \rightarrow 0$	$0.5 \rightarrow 1$	1
1	1	0	0	$-0.5 \rightarrow 0$	0

Back Propogation Training Algorithm for FFNN Network for XOR

→ Refine Weight values



→ N/w Activation Forward Pass
--> Error Propogation Backward Pass

Weight Update Rule

$$w_{ji} = w_{ji} + \Delta w_{ji}$$

$$\Delta w_{ji} = n * [\delta E / \delta w_{ji}]$$

n is learning in [0.1, 0.9]

Stopping Criterion

→ Total mean squared error change

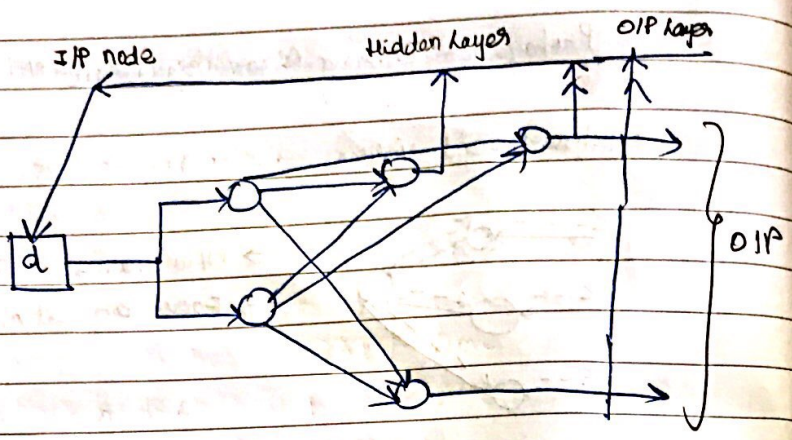
→ Generalization based ~~co~~ criterion

Delta Rule for Error Minimization

$$E_{av} = \sum_{i=1}^{N} \Sigma(i) N$$

→ Recurrent Network

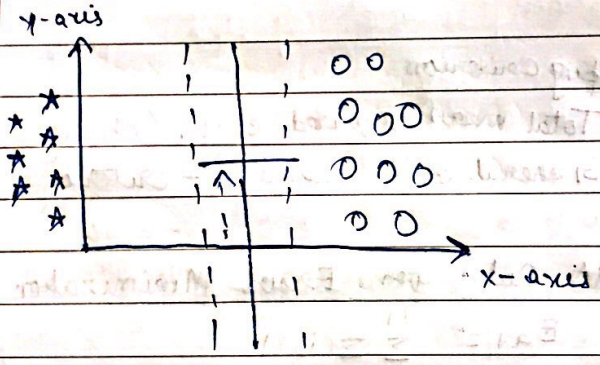
- Non Cycle Graph : Backpropogation
- Recurrent : Form a cycle
- In recurrent network, output will be given as i/p
- In backpropogation, error value will be backpropogated



→ Support Vector Machine

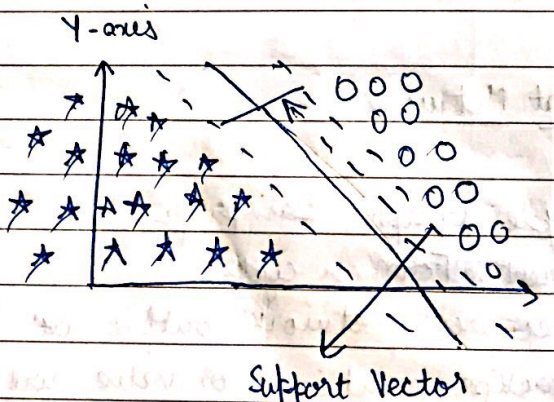
- Linearly Separable and Non Linearly Separable Functions.

(i)



Case 1:
Small Margin

(ii)

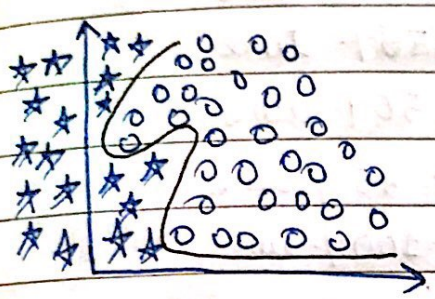


Case 2:
Line Margin

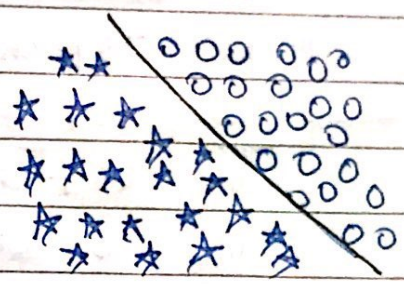
→ In non linearly separable, we use Kernal functions to separate data.

Transformation into higher dimensional space
- we use kernal function to linearly separate classes. We use higher degree polynomial

Input Space



Feature Space



18-03-19

Learning Probabilistic Models

→ Two flavours are present for a chocolate, cherry and lime, which have the same wrapper.

⟶ Hypotheses:

h1: 100% cherry + 0% lime
h2: 75% cherry + 25% lime
h3: 50% cherry + 50% lime
h4: 25% cherry + 75% lime
h5: 0% cherry + 100% lime

Probability of each hypothesis

$$P(h_i \mid d) = \alpha P(d \mid h_i) P(h_i)$$

To Predict Lime / Cherry:

$$P(X \mid d) = \Sigma P(X \mid d, h_i) P(h_i \mid d)$$
$$= \Sigma P(X \mid h_i) P(h_i \mid d)$$

→ Learning with Complete Data

- Agent tries to know by using the given complete data.
- We do not have any hypothesis here.
- Likelihood of particular dataset:

$$P(d \mid h\theta) = \prod_{j=1}^{N} P(d_j \mid h\theta) = \theta^c (1-\theta)^l$$

- Maximizing log likelihood

$$L(d \mid h\theta) = \log \Pi P(d \mid h\theta) = \sum_{j=1}^{N} \log(d_j \mid h\theta) =$$
$$c \log\theta + l \log(1-\theta)$$

- Set resulting expression to '0'

$$\frac{dl(d(h\theta))}{d\theta} = \frac{c}{\theta} - \frac{l}{1-\theta} = 0$$

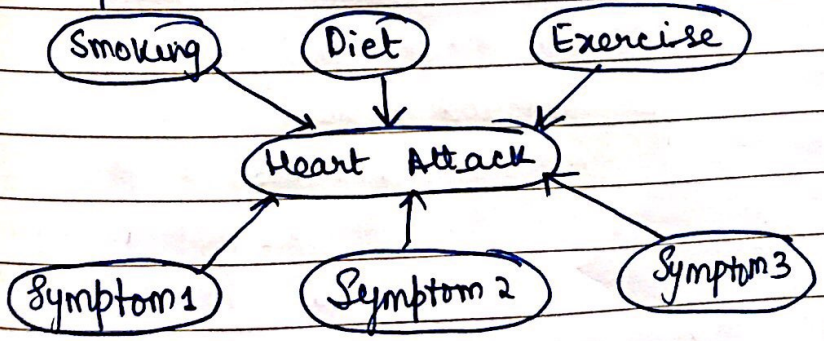
$$\Rightarrow \theta = \frac{c}{c+l} = \frac{c}{N}$$

- Naive Bayes Model

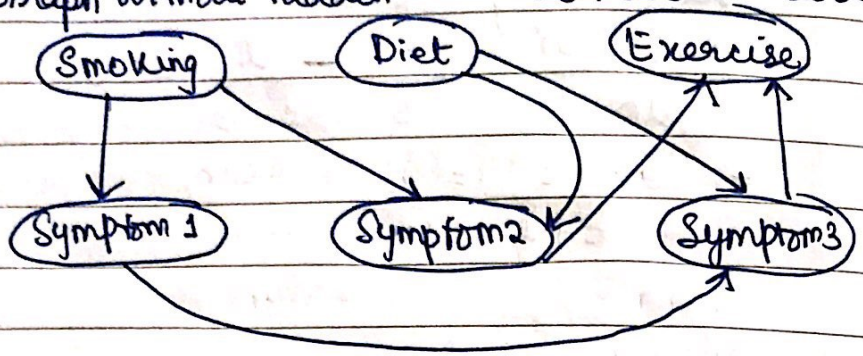
$$P(C|x_1 \cdots x_n) = \alpha P(c) \pi P(x_i|c)$$

Learning with Hidden Variables

- Missing hidden variables arises complications
- Links will become more.
- Eg - Heart Attack is caused due to smoking, drinking, etc.
  Remove hidden variable ~~smoke~~ heart attack.
  Links will be more, smoking and drinking may cause other things.
- Graph will become more complex if we remove the hidden variables.
- Graph with hidden variable heart attack:



- Graph without hidden variable Heart Attack.



→ **EM Algorithm**

Here:

x = observed value

z = hidden variables

θ = all the parameters

E-Step: Computation of Summation

M-Step: Maximization of log likelihood

$$\theta^{(i+1)} = \underset{\theta}{\operatorname{argmax}} \sum_{z} P(z = z \mid x, \theta^{(i)}) \, L(x, z = z \mid \theta)$$

NATURAL LANGUAGE PROCESSING

UNIT-V: NATURAL LANGUAGE FOR COMMUNICATION

Natural Language Processing

- How the machine/agent learns English.
- Language is a set of strings.
- P(S = words) S = sentence
- N-gram character model.
  - Model of probability distribution of n-letter sequence.
  - Defined as Markov Chain, depends on immediate previous state.
  - Trigram Model (three states):
    $P(c_i | c_{1:i-1}) = P(c_i | c_{i-2:i-1})$
  - Probability of Sequence of Characters
    $P(c_{1:N}) = \prod_{i=1}^{N} P(c_i | c_{1:i-1}) = \prod_{i=1}^{N} (P(c_i | c_{i-2:i-1}))$
  - N-gram model is widely used in the language identification.
  - Initially, we must build trigram character model for identification of language:
    $P(c_i | c_{i-2:i-1}, l)$
  - Corpus: Body of text
  - After building trigram model, we divide it into training model and evaluation model, and give it a dataset that it has not seen.
  - After building trigram model, we get the probability of text from language and we get the most probable language by using the Baye's rule and also Markov Assumption.

P(text | language)

$$l^* = \underset{l}{\operatorname{argmax}}\ P(l \mid c_{1:N})$$

$$= \underset{l}{\operatorname{argmax}}\ P(l)P(c_{1:N} \mid l)$$

$$= \underset{l}{\operatorname{argmax}}\ P(l) \prod_{i=1}^{N} (c_i \mid c_{i-1:i-1}, l)$$

- Smoothing N-gram model:
  - Word starting with:
    'th' → frequent
    'ht' → unknown
  - Even if we do not train that dataset, it should understand and give the correct output.
  - Process of adjusting the probability of low frequency count is smoothing.
  - Laplace Smoothing: Poor performance, but it goes in the right direction.
  - Back-Off-Model
    → Better Method
    → Starts counting / estimating 'n' gram counts.
    → Linear Interpolation Smoothing is a backoff model that combines unigram, bigram and trigram.

$$\hat{P}(c_i \mid c_{i-2:i-1}) = \lambda_3 P(c_i \mid c_{i-2:i-1}) + \lambda_2 P(c_i \mid c_{i-1}) + \lambda_1 P(c_i)$$

$$\lambda_3 + \lambda_2 + \lambda_1 = 1$$

→ λ value can be fixed or varied

If unigram / bigram / trigram value is high, we increase λ at that gram and decrease at the other

- Model Evaluation
  - How a model is built and how are we evaluating it.
  - Perplexity describes the probability of a sequence.
  - We must build a training and testing model, and give unknown dataset to estimate accuracy
  - Perplexity $(c_1 : N) = P(c_1 : N)^{-1/N}$
  - Perplexity is calculated for:
    unigram → more mistakes
    bigram → less grammar mistakes
    trigram → perfect sentence

- N-gram word models
  - For an unknown word in the vocabulary, we add it as: <UNK>