Module 2

Supervised Learning - Classification

Machine Learning Models

Discriminative Generative

1. Discriminative Model

- makes prediction based on conditional probability.
- used for classification or regression.
- also known as conditional model.
- Learns boundaries between classes or labels in a dataset
- the goal is to find the decision boundary separating one class from another.

ex discriminating students in high school
1) will go to college
2) will discontinue studies
3) will go to trade school

Some pattern must be there that decided where a student belong to after school. We collect family background, academic info to make the decision.

- focuses on predicting labels of the data.

Types/Examples
1) Logistic Regression
2) Support vector machine
3) Decision Tree
4) Random forest

2. Generative Model

- focuses on distribution of individual classes in a dataset and the learning algorithms tend to model the underlying patterns / distributions of the data points
- uses joint probability
- uses probability estimates & likelihoods to model data points
- capable of generating new data instances

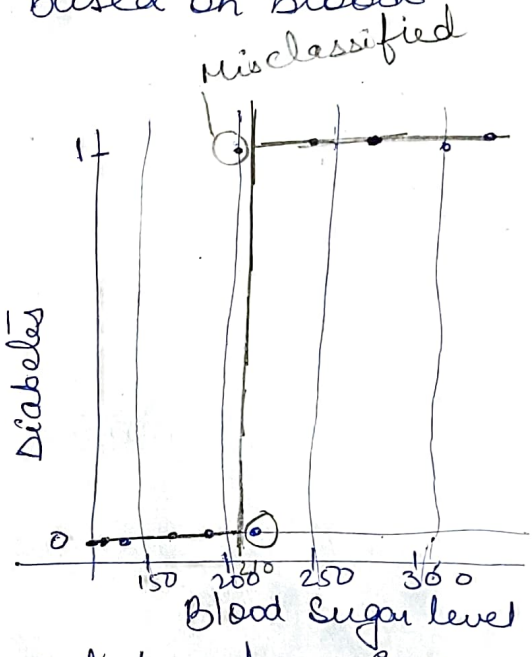
Types / Examples

1. Naive Bayes
2. LDA
3. Hidden Markov model

Logistic Regression (solves binary classification problems)

ex Diabetes - person is diabetic or not based on blood sugar level.

Blood Sugar level	Diabetes
190	0
240	1
300	1
160	0
200	1
269	1
129	0
141	0
220	0
337	1



We can decide a person is diabetic or not by drawing a decision boundary

Suppose we draw a decision boundary at 210. All patients whose BSL < 210 are non-diabetic & above 210 are diabetic

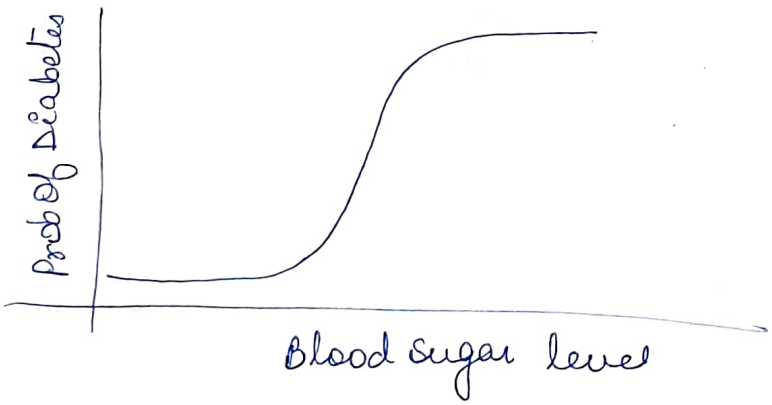
→ Misclassification problem - deciding class based on cutoff seems risky. Talk in terms of probability.

Overcome - Sigmoid Curve

↓

models the probability

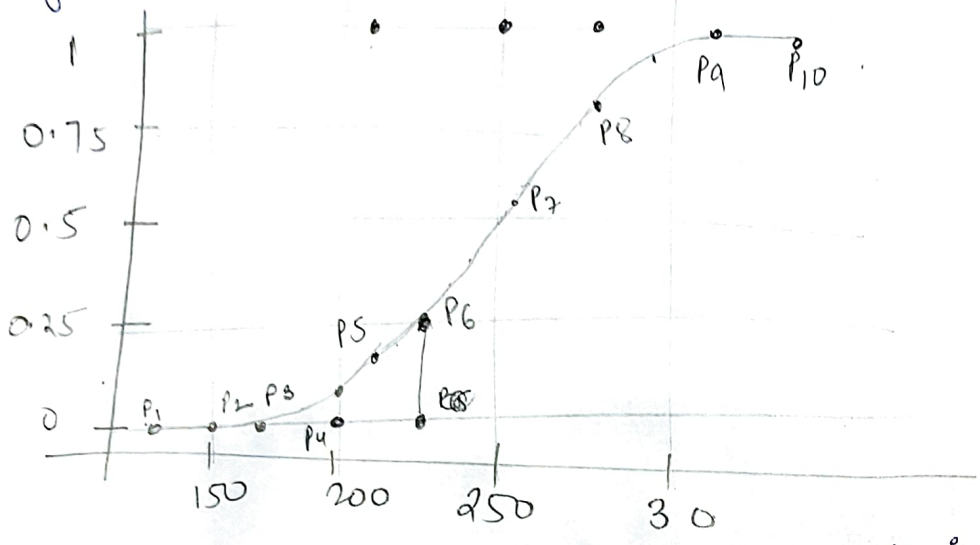
$$P(\text{probability of Diabetes}) = \frac{1}{1+e^{-(\beta_0+\beta_1 x)}}$$



+ has extremely low values in start, extremely high levels in the end & intermediate values in the middle

Likelihood

→ finding best fit curve.
→ vary values of β_0 and β_1 until we get the combination of beta values that maximises the likelihood.



$\beta_0 = -15$
$\beta_1 = 0.065$

P_1, P_2, P_3, P_4, P_6 to be make small, rest maximise

$$\text{Likelihood} = (1-P_1)(1-P_2)(1-P_3)(1-P_4)(1-P_6)(P_5)(P_7)(P_8)(P_9)(P_{10})$$

* The best fitting combination of β_0 and β_1 will be the one that will maximise the product.

— This process where we vary the betas until we find the best fit curve for probability of diabetes is called logistic regression.

Model Evaluation : Accuracy, Sensitivity and Specificity

Confusion Matrix

Actual	Predicted: No	Predicted: Yes
No	True Negatives	False Positives
Yes	False Negatives	True Positives

$$\text{Accuracy} = \frac{\text{Correctly Predicted labels}}{\text{Total No. of labels}}$$

$$\text{Sensitivity} = \frac{\text{True Positives}}{TP + FN}$$

$$\text{Specificity} = \frac{\text{True Negative}}{TN + FP}.$$

Multivariate Logistic Regression

$$P = \frac{1}{1+e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots \beta_n x_n)}}$$

Naive Bayes

Probability - chances of occurrence of an event ; ratio of number of desired outcomes to the number of total possible outcomes.

$$P(E) = \frac{\text{Favourable outcomes}}{\text{Total outcomes}}$$

Conditional Probability

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Joint Probability

$$P(A \cap B) = P(A|B) * P(B)$$

ex

	India win	India Lose	Total
B = Century — Sachin Scores a Century	10	2	12
B = No century — Sachin doesn't score a Century	50	38	88
Total	60	40	100
	A = win	A = loss	

Prob of India win $\leftarrow P(A) = \frac{60}{100}$

Prob of Sachin scoring century $P(B) = \frac{12}{100}$

~~Prob of India lose~~ Conditional Probability

(Prob of India wins when sachin scored century)

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{10/100}{12/100} = \frac{10}{12}$$

Bayes Theorem

$$P(A \cap B) = P(A|B) \cdot P(B)$$
$$P(A \cap B) = P(B|A) \cdot P(A)$$

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

$$\boxed{P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}}$$

Naive Bayes with One Feature

Sno.	Type of Mushroom	Cap Shape
1	Poisonous	Convex
2	Edible	Convex
3	Poisonous	Convex
4	Edible	convex
5	Edible	convex
6	Poisonous	convex
7	Edible	Bell
8	Edible	Bell
9	Edible	Convex
10	Poisonous	convex
11	Edible	Flat
12	Edible	Bell

Type of mushroom is target variable

~~P(C~~

~~P(X = convex~~

$P(C = \text{edible} | X = \text{Convex})$

$P(C = \text{poisonous} | X = \text{Convex})$

Naive Bayes for Categorical Data -

$$P(C_i|X) = \frac{P(X|C_i) * P(C_i)}{P(X)}$$

C_i denotes classes
X denotes features of the data point

The effect of denominator $P(X)$ is not incorporated while calculating prob as it is a scaling factor.

Naive Bayes follows an assumption that the variables are conditionally independent given the class.

$P(C_i)$ is prior prob.
It is the prob of an event occuring before collection of new data.

Naive Bayes with multiple columns

Type of Mushroom	Cap-shape	Cap-surface
		Smooth
		Smooth
		Scaly
		Scaly
		Smooth
		Smooth
		Fibrous
		Scaly
		Smooth
		Scaly
		Fibrous
		Scaly
		Fibrous
		Scaly

Q. Classify a new mushroom whose cap shape is convex & cap-surface is smooth.

$$P(\text{edible} = \text{yes} \mid x = (\text{convex}, \text{smooth})) =$$

$$P(x = \text{convex} \mid \text{edible}) * P(x = \text{smooth} \mid \text{convex}) * P(\text{edible})$$

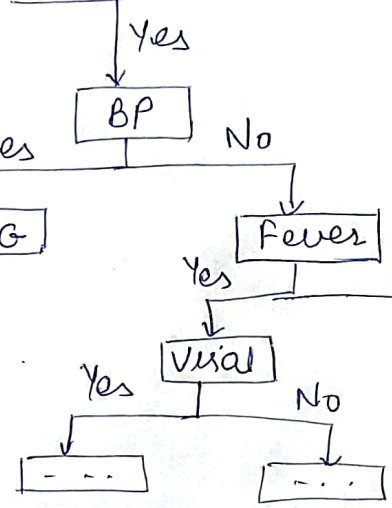
$$P(\text{edible} = \text{no} \mid x = (\text{convex}, \text{smooth})) =$$

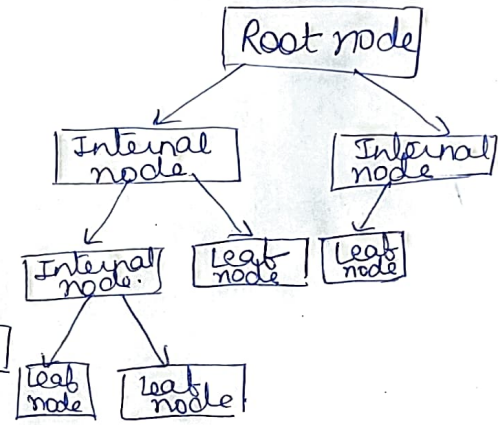
$$P(x = \text{convex} \mid \text{poisonous}) * P(x = \text{smooth} \mid \text{poisonous}) * P(\text{edible})$$

* effect of denominator is unaffected so removed.

Decision Trees

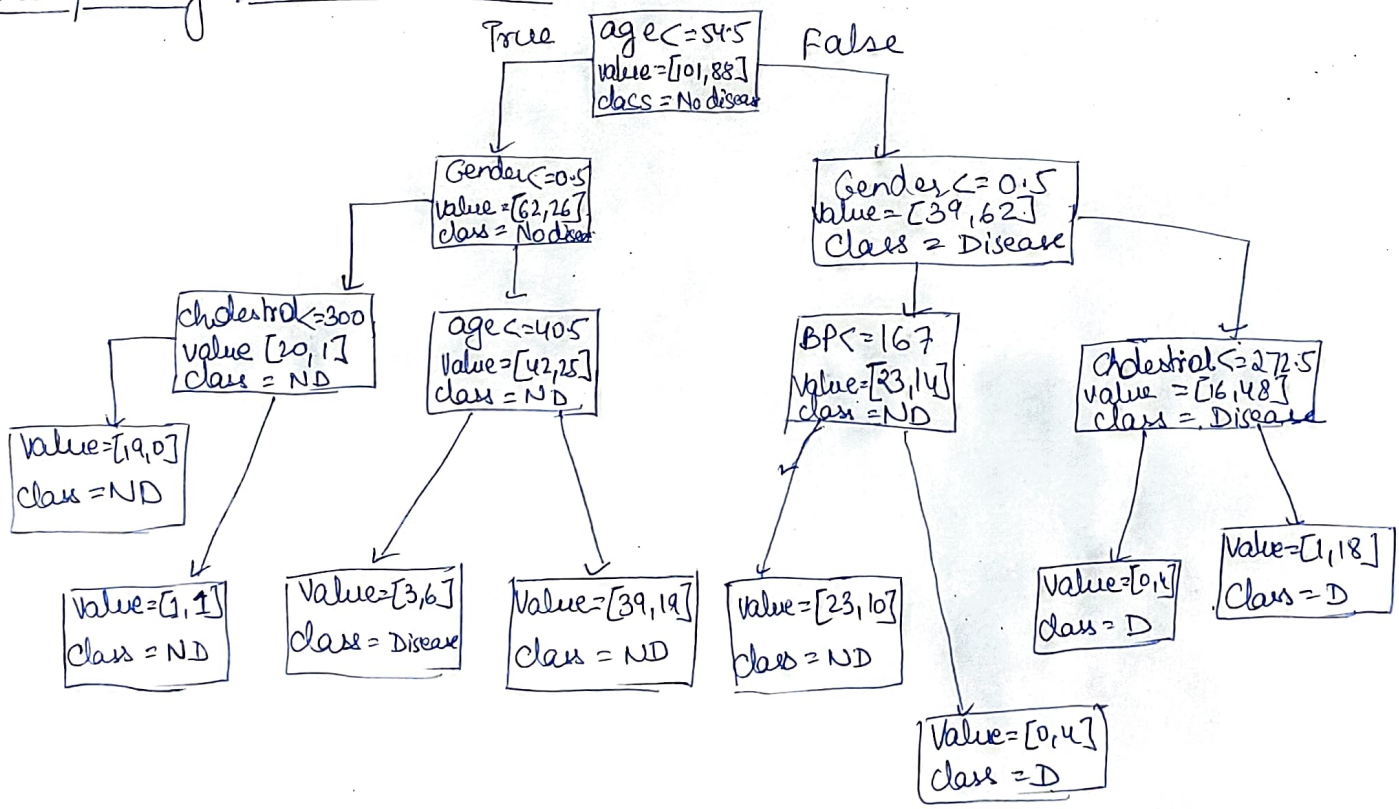
→ models highly non-linear data.
→ is a tree-like model to make predictions.
→ splits data into multiple sets of data. Each of these sets is then further split into subsets to arrive at a decision
→ uses decision-making process → asks series of questions in a nested if-then else structure. On each node, question is asked & further the data is split which is held by the node.
→ highly interpretable model.





* Root node contains all the examples / instances

Interpreting Decision Trees



Building Decision Trees

1. Recursive binary splitting the data into smaller subsets
2. Select best variable for the split
3. Applying the split based on the rules obtained from the attributes.
4. Repeating the process for the subsets obtained
5. Continuing the process until stopping criterion is reached
6. Assigning the majority class / average value as the prediction.

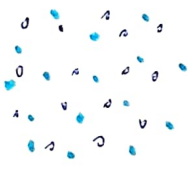
* top-down approach
* greedy approach

Advantages of Tree Models Over Linear Models

1. Predictions are easily & interpretable.
2. Versatile:- can handle both classification & regression
3. Fast:- partitioning works well with large data sets.
4. Handle multicollinearity better
5. Scale-invariant
6. Can identify complex relationships :- works well in cases where we can not fit a single linear relationship between the target & feature variables.

Splitting and Homogeneity

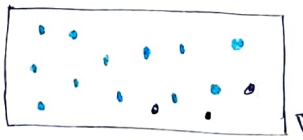
→ More the homogeneity, lesser is the variance, better will be the split



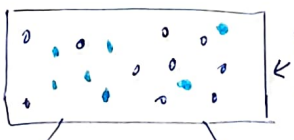
50-50 Non Homogeneous

Completely Homogeneous

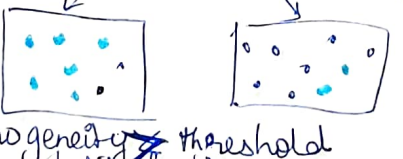
* split the data set such that the homogeneity of the resultant partition is maximum.



Homogeneity > Threshold
No split further



Homogeneity < Threshold
split further



Homogeneity > Threshold
no split further

How to measure Homogeneity (or Impurity)

1. Classification error
2. Gini Index
3. entropy

1. Classification error

$$E = 1 - \max(P_i)$$

Class	0 : 20
	1 : 80

$$P_0 = \frac{20}{20+80} = 0.2 \qquad P_1 = \frac{80}{100} = 0.8$$

$$E = 1 - P_{max}$$
$$= 1 - 0.8 = 0.2$$

If we assign everything to majority class, what is the error rate.

2. Gini Impurity

$$G = \sum_{i=1}^{k} P_i(1-P_i)$$

$$G = P_0(1-P_0) + P_1(1-P_1) = 0.2 \times 0.8 + 0.8 \times 0.2 = 0.32$$

3. Entropy

$$D = -\sum_{i=1}^{k} P_i \log P_i = -0.2 \log(0.2) - 0.8 \log(0.8)$$
$$= 0.72$$

Gini index is the degree of a randomly chosen datapoint being classified incorrectly. A Gini Index 0 means that all data points belong to a single class

* Higher the homogeneity, lower the Gini Index

Entropy is the degree of disorder in the given data. value ranges from 0 to 1.

* Higher the homogeneity, lower the entropy

How do we identify the attributes that results in the best split?

Algorithms

1. CART 2. C4.5 3. C5.0 4. ID3 5. CHAID

change in impurity or purity gain

Δ Impurity = Impurity (pre-split) − Impurity (post-split)

ex No disease : 60 (class 0)
Disease : 40 (class 1)

$P_0 = \frac{60}{60+40} = 0.6$ $P_1 = \frac{40}{60+40} = 0.4$

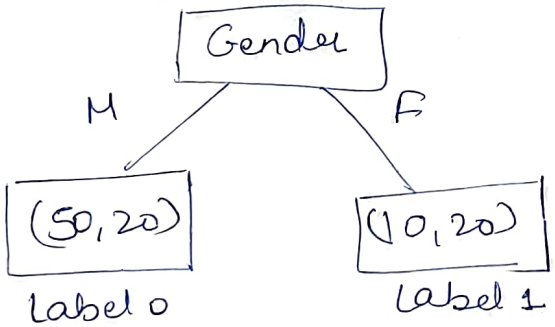
Gini impurity before split

$P_0(1-P_0) + P_1(1-P_1) = 0.6 \times 0.4 + 0.4 \times 0.6 = 0.48$

Feature	Gender		
	M	F	Total
No disease	50	10	60
Disease	20	20	40
Total	70	30	100

Feature	Cholesterol		Total
	<250	>250	
No disease	50	10	60
Disease	10	30	40
Total	60	40	100

1) Cal homogeneity reduction on attribute gender.



Males

$P_0 = \frac{50}{70} = 0.714$ $P_1 = \frac{20}{70} = 0.286$

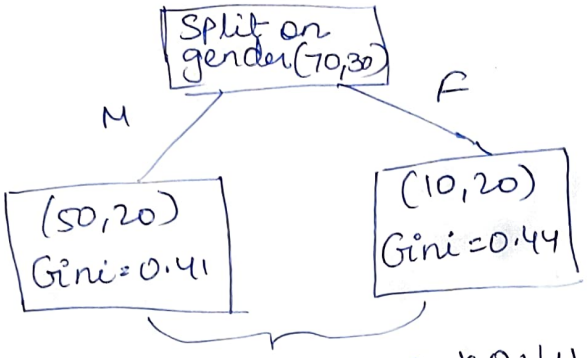
Gini impurity (for males) $= 0.714(1-0.714) + 0.286(1-0.286) = 0.41$

Females $P_0 = \frac{10}{30} = 0.333$ $P_1 = \frac{20}{30} = 0.667$

$G = 0.333(1-0.333) + 0.667(1-0.667) = 0.44$

$P_{male} = \frac{70}{100} = 0.7$ $P_{female} = \frac{30}{100} = 0.3$

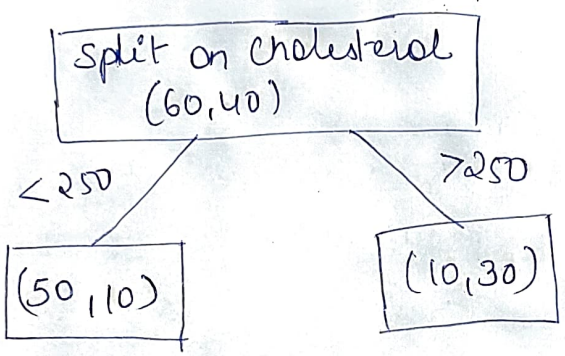
Gini impurity after split $= 0.7 \times 0.41 + 0.3 \times 0.44 = 0.42$



$$\text{Overall Gini} = \frac{70}{100} \times 0.41 + \frac{30}{100} \times 0.44 = 0.42$$

$$\Delta \text{Gini}_{\text{sex}} = 0.48 - 0.42 = 0.06$$

2) Cholesterol



$$\Delta \text{Gini}_{\text{Cholesterol}} = 0.48 - 0.3 = 0.18.$$

* there is higher reduction in Gini impurity when we split the dataset on cholesterol as compare to gender.

Disadvantages of Decision Tree

1. Strong tendency of Decision tree to overfit the data. If allowed to grow with no check on its complexity, a decision tree will keep splitting until it has correctly classified all the data points in the training set.
2. They tend to be quite unstable, which is an implication of overfitting. A few changes in the data can considerably change a tree.

Two strategies to control Overfitting in decision trees :

1) Truncation
2) Pruning

1) Truncation/Pre-Pruning -

Stop the tree while it is still growing so that it may not end up with leaves containing very few data points.

Methods of Truncation

1. Limit the minimum size of a partition after a split

 → Enough samples for split

 → Not enough samples, do not split

2. If homogeneity measure do not change after we split on a particular attribute, then do not split further & make that node as leaf node.
3. Limit depth of the tree.
4. Set a limit on maximum number of leaves present in tree.
5. Set a minimum threshold on the number of samples that appear in a leaf.

(180, 3)

2) Pruning -

- Method of chopping off parts of a tree once it is fully grown.
- Bottom up approach
- Look at non leaf nodes & chop off

If the accuracy of pruned tree is higher than accuracy of original tree then chop that branch

Before Pruning

→ look at data at this node & try merging the info in a single leaf node

* This is a regularization step.

After Pruning

Adv - decrease in tree complexity & also helps in reducing Overfitting

Ensembles

→ refers to a groups of things viewed as a whole rather than individually.

→ in an ensemble, a collection of models is used to make predictions rather than individual models.

→ Random forest is an ensemble made by the combination of a large number of decision trees.

→ an ensemble can be made by combining all types of models. An ensemble can have a logistic regression, neural n/w & few decision trees working in unison.

→ The idea in creating an ensemble is to make sure that we have a collection of models which are diverse and acceptable models

diverse → ensures that models are complementary to each other & individual models make predictions independent of each other.

acceptable → each model is at least better than a random model.

ways to achieve diversity

1) Different subsets of training data given to different models. (Same model but providing diff training data we can have 4 eg logistic reg)
2) Providing different training hyperparameters.
3) Use different types of classifiers - instead of using only decision tree or logistic reg, combine different models
4) Use different features to different models.

* Majority score

Some popular ensembling approaches are boosting, bagging, voting, stacking, blending

1. Bagging

creates different training subsets from the sample training data with replacement & an algorithm with the same set of hyperparameters is built on these different parts of data resulting in slight difference b/w the individual models.

The predictions of these individual models are combined by taking the average of all the values for regression or a majority vote for classification problems.

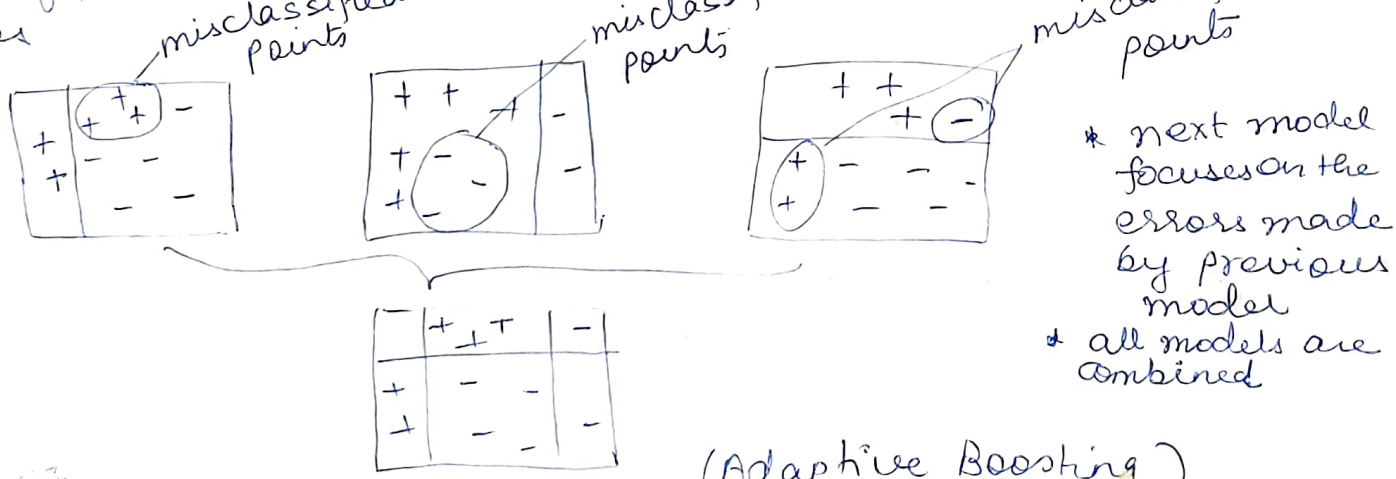
→ works well with high variance algorithms like decision trees.

→ disadvantage is that we cannot really see the individual trees one by one & figure out what is going on behind the ensemble as it is a combination of n number of trees working together.

Another disadvantage is that if any feature dominates then all the trees will look similar & hence the proper of diversity in ensembles is lost

2. Boosting

most popular approaches to ensembling. It can be used with any technique and combines weak learners into strong learners by creating sequential models such that the final model has high accuracy than the individual models



(Adaptive Boosting)

→ Creates sequential models
→ Combines weak learners into stronger ones
→ individual models are connected in such a way that they correct the mistakes made by the previous models. (weak learners)
→ any model can be a weak learner - linear regression Decision Tree. For ex decision tree of depth 1 (decision stump)
→ the adv of weak learners is that they don't overfit.
→ a weak learner is better than a random guesser.
→ approaches to Boosting - Adaptive & Gradient.

Random Forest

→ ensemble of a number of decision trees.
→ Builds 100's of decision trees on different samples from the data set;
→ Takes majority score

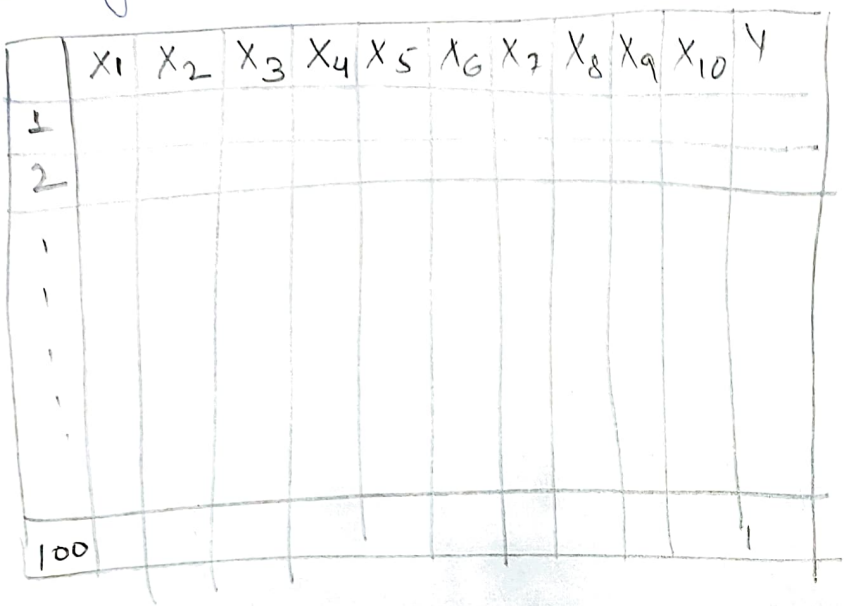
Bagging is an acronym of -
- Bootstrapping - building models on random samples from input data.
- Aggregation - aggregate results from bootstrapped models using majority score.

Decision Tree Recap

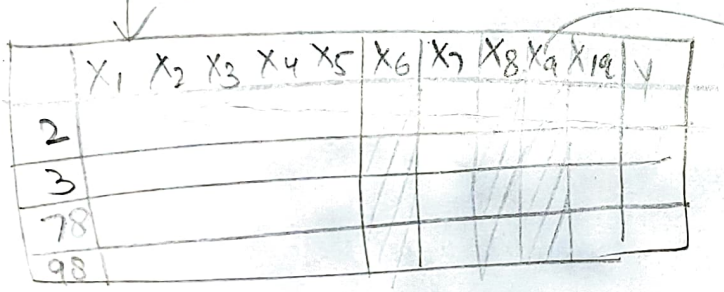
1. Assumptions
(i) Data is in tabular format
(ii) Each row represents a data point
(iii) Each column represents an attribute
2. Building the Decision Tree
(i) Node represents test on an attribute
(ii) Leaves contain the predictions for the target column.
(iii) For every split at the node, training data homogeneity should increase

(iv) we split on attributes which cause maximum increase in homogeneity.

Building a Random forest

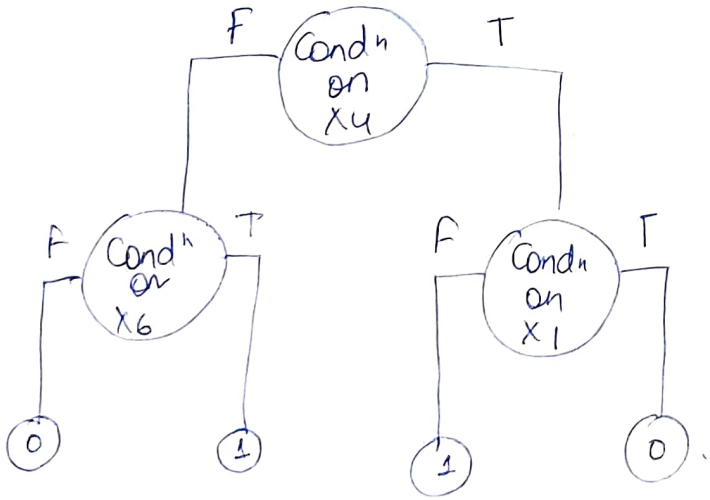


Bootstrapped Sample for building the Decision Tree



* The attributes that it considers for the split is a subset of the complete set of attributes

* Bootstrapped sample
* Subset of attributes at each split



Neural Networks (Module 2)

Artificial Neural N/w is inspired by the human brain
Human brain is made up of neurons
ANN is a collection of many simple devices called artificial neurons. The n/w learns to conduct certain tasks such as recognising an object by training the neurons to fire in a certain way when given a particular i/p

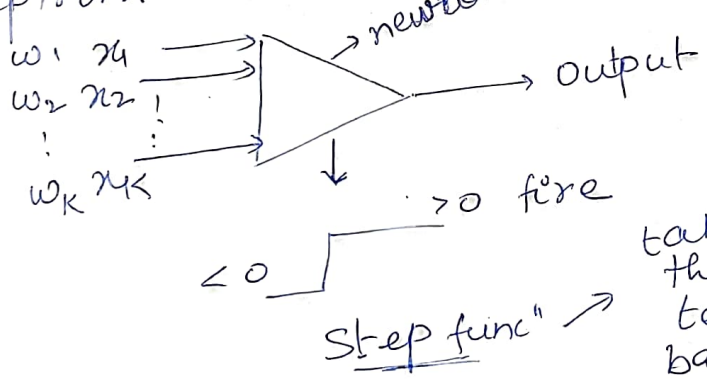
Perceptrons

Consider a sushi place you plan to visit this saturday There are various factors that would affect this decision such as:

1) The distance b/w the sushi place & your home
2) The cost of the food they serve there
3) The number of people accompanying you

You make such a decision based on multiple such factors. Also each decision factor has a different weight eg the dist of the place might be more imp than the no. of people accompanying you.

Perceptrons work in a similar manner.



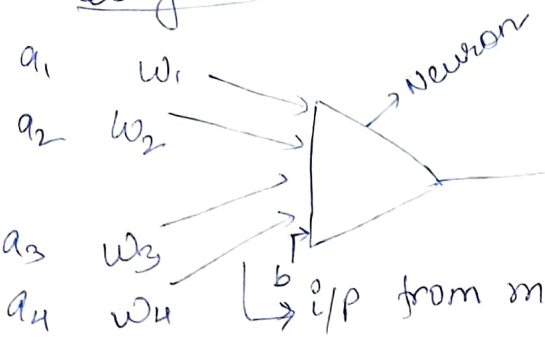
Step func" → takes weighted sum of the i/ps. & a constant bias term & takes decision based on some threshold.

For ex if weighted sum is > 0 then the neuron will fire else not.

* perceptron takes a weighted sum of multiple i/p (with bias) as the cumulative i/p & applies an o/p func on the cumulative i/p to get the o/p which then assists in making a decision.

$$\text{Cumulative Input} = w_1 x_1 + w_2 x_2 + \dots w_n x_n + b$$

Single Neuron

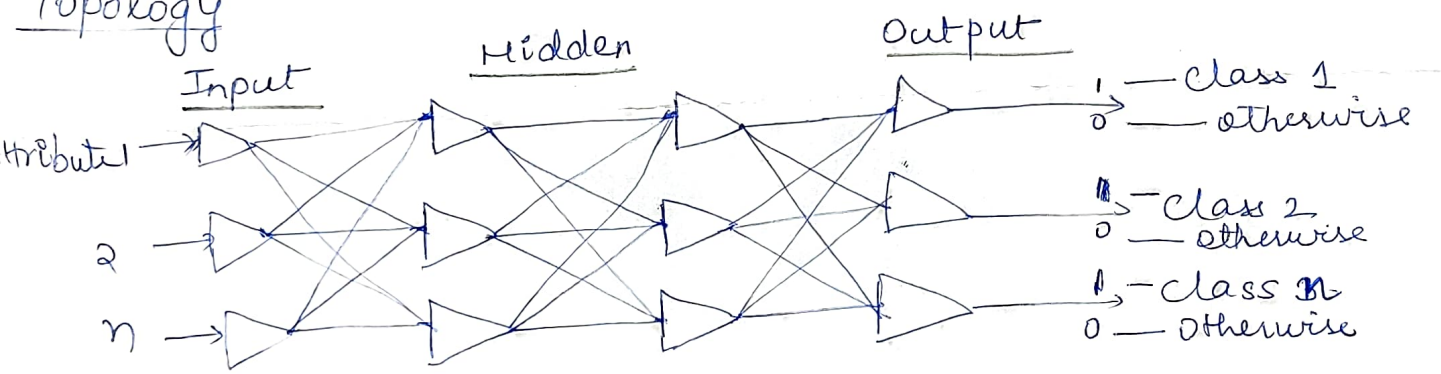


Inhibitors Analogy – these weights will either amplify the weights coming from other neurons or suppress it. If we want to block out the weight coming from some other neuron we just have to make that weight 0.

Activation func^n – transforms the cumulative i/p & results in activation of the neuron.

Activation function $f(a_1w_1 + a_2w_2 + a_3w_3 + a_4w_4)$

Topology



Takes i/p from outside world

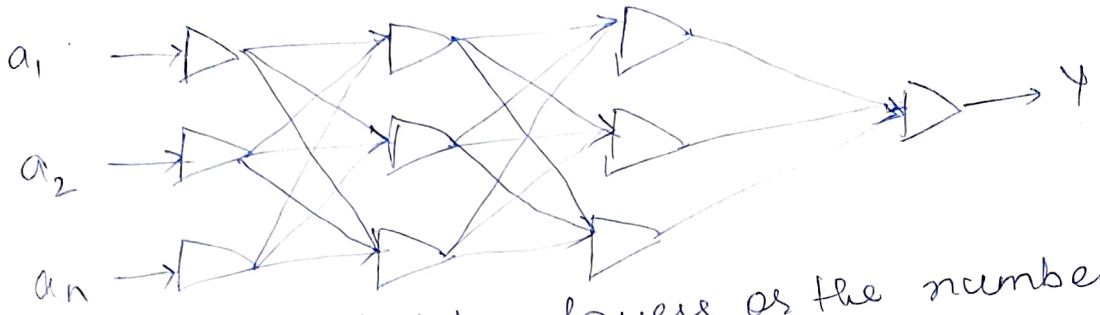
Classification

Multiple artificial neurons in a neural n/w are arranged in different layers. The first layer is known as input layer & the last layer is called the output layer. The layers in between these two are hidden layers.

The number of neurons in the input layer is equal to the number of attributes in the data set & the no. of neurons in the o/p layer is determined by the no. of classes of the target variable –

For classification problem

- For a regression problem, the number of neurons in the output layer would be 1 (numeric variable)



* The no. of hidden layers as the number of neurons in each hidden layer or the activation functions used in the neural n/w changes acc. to the problem & these details determine the topology or structure of the neural n/w.

∴ There are six main elements for any neural network
1) Input layer 2) output layer 3) Hidden layers
4) Weights & biases 5) Activation functions
6) Structure < nodes - Neuron
Edges - interconnection

* The inputs can only be numeric, for different types of input data, we need to use different ways to convert the inputs into a numeric form.
For text data we can use one-hot vector or word embeddings
Images can be represented as arrays of numbers

Depending on the nature of the given task, the outputs of neural networks can be either in the form of classes or numeric

One of the commonly used output functions is the Softmax function for classification
Similar to multivariate logistic regression

$$P_i = \frac{e^{w_i \cdot x'}}{\sum_{t=0}^{C-1} e^{w_t \cdot x'}}$$

C = no. of classes or neurons in o/p layer
x' = i/p to n/w
w_i = weights

(exponential can never be negative)
so o/p produced by neuron will always positive

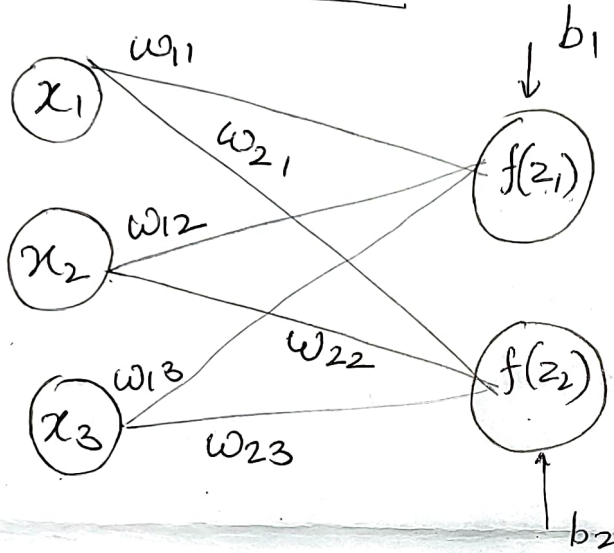
Feedforward Neural Network

Module 2

In ANN the O/P from one layer is used as input to the next layer, such networks are called feedforward neural networks

There are no loops in the network i.e information is always fed forward, never fed backward.

Matrix Representation



$$z_1 = w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + b_1$$

$$z_2 = w_{21}x_1 + w_{22}x_2 + w_{23}x_3 + b_2$$

$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

The activation function is applied to each element of the vector.

$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \sigma(W x_i + b) = \begin{bmatrix} \sigma(w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + b_1 \\ \sigma(w_{21}x_1 + w_{22}x_2 + w_{23}x_3 + b_2 \end{bmatrix}$$

Problem Statement - To predict the price of a house, given the size of the house and the number of rooms available

Std No. Of Rooms	Std. House Size	Price
3	1340	313000
5	3650	2384000
3	1930	342000
3	2000	420000
4	1940	550000
2	880	490000

Pseudocode

1. $h^0 = x_i$ — Initialize h^0 as input
2. for l in $[1, 2, \ldots, L]$: → for each layer
   $h^l = \sigma(w^l . h^{l-1} + b^l)$ → w^l: weight of current layer
   h^{l-1} : o/p of previous layer
   b^l : bias of current layer
3. $p = f(h^l)$ → compute prediction p by applying an activation function to the o/p from previous layer.

In both regression & classification problems, the same algorithm is used till last step. In the final step, in the classification problem, p defines probability vector which gives the probability of the data point belonging to a particular class among different possible classes or categories.

In regression problem, p represents the predicted output obtained.

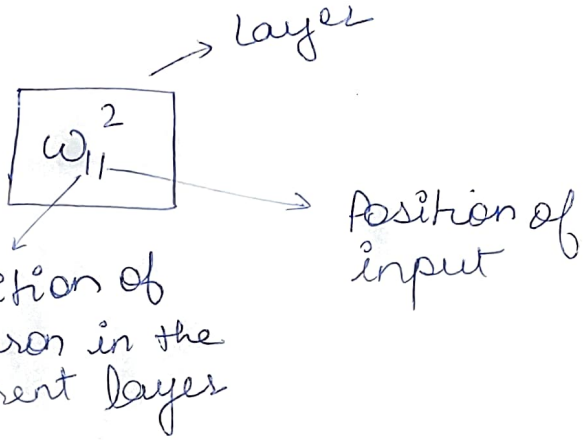
For classification problem we use softmax output

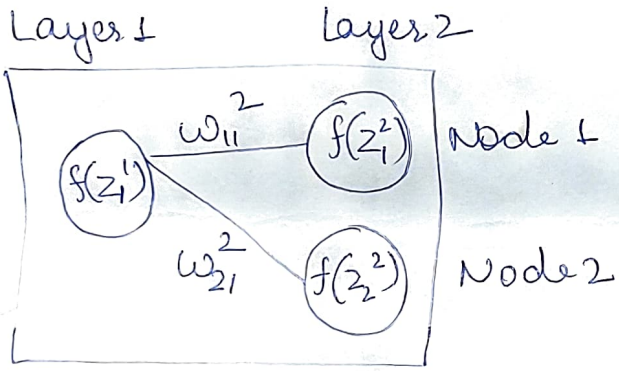
$$P_{ij} = \frac{e^{w_j h^L}}{\sum_{t=1}^{c} w_t h^L} \qquad j = [1, 2, \ldots c]$$

P_{ij} is often called normalising the vector p

So complete feedforward algo for classification problem

1. $h^0 = x_i$
2. For l in $[1, 2, \ldots L]$: $h^l = \sigma(w^l h^{l-1} + b^l)$
3. $p_i = e^{w^0 h^L}$
4. p_i = normalise (p_i)





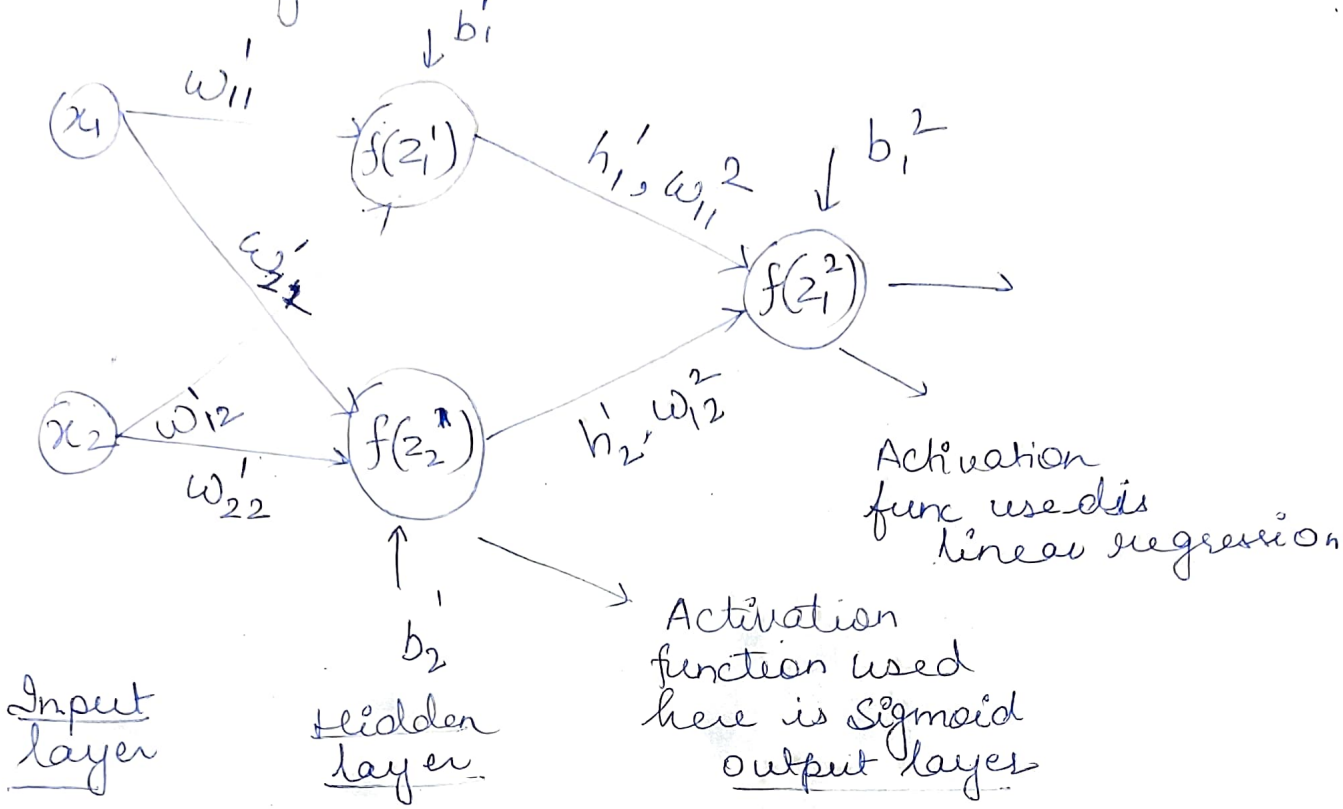
Numerical

① Scaling is done for std. Number of Rooms & std. House Size & log transformation is applied on price column

Scaling → ((obs - mean)/std. deviation)

Std no. of Rooms	std. House Size	Price
-0.32	-0.66	-0.54
1.61	1.80	2.03
-0.32	-0.03	-0.51
-0.32	0.05	-0.41
0.65	-0.02	-0.25
-1.29	-1.15	-0.32

We will be using this architecture



1. Layer 1 : Node 1

Computing the cumulative input for the first node of the hidden layer;

$$z_1^1 = w_{11}^1 x_1 + w_{12}^1 x_2 + b_1^1$$

$$= 0.2 \times (-0.32) + 0.15 \times (-0.66) + 0.1 = -0.063$$

Output from first node $h_1^1 = \sigma(-0.063) = \frac{1}{1+e^{-\varepsilon}} = \frac{1}{1+e^{-(0.063)}} = .48$

2. Layer 2 ; Node 2

$$z_2^1 = w_{21}^1 x_1 + w_{22}^1 x_2 + b_2^1$$

$$= 0.5 \times (-0.32) + 0.6 \times (-0.66) + 0.25 = -0.306$$

Output from second node;

$$h_2^1 = \sigma(-0.306) = \frac{1}{1+e^{-\varepsilon}} = \frac{1}{1+e^{-(-0.306)}} = 0.424$$

Matrix Representation

$$x^1 = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -0.32 \\ -0.66 \end{bmatrix}$$

$$w^1 = \begin{bmatrix} w_{11}^1 & w_{12}^1 \\ w_{21}^1 & w_{22}^1 \end{bmatrix} = \begin{bmatrix} 0.2 & 0.15 \\ 0.5 & 0.6 \end{bmatrix}$$

$$b^1 = \begin{bmatrix} b_1^1 \\ b_2^1 \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.25 \end{bmatrix}$$

$$h^1 = \begin{bmatrix} h_1^1 \\ h_2^1 \end{bmatrix} = \sigma(w^1 x^1 + b^1) = \sigma\left(\begin{bmatrix} w_{11}^1 & w_{12}^1 \\ w_{21}^1 & w_{22}^1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1^1 \\ b_2^1 \end{bmatrix}\right)$$

$$= \sigma\left(\begin{bmatrix} w_{11}^1 x_1 + w_{12}^1 x_2 + b_1^1 \\ w_{21}^1 x_1 + w_{22}^1 x_2 + b_2^1 \end{bmatrix}\right)$$

Layer 2: Node 1

Moving on to the output layer with a linear activation function:

$$Z_1^2 = b_1^2 + w_{11}^2 h_1^1 + w_{12}^2 h_2^1$$
$$= 0.4 \times 1 + 0.3 \times 0.484 + 0.2 \times 0.424$$
$$= 0.63$$

Since the activation function is linear for this regression problem

$$h_1^2 = Z_1^2 = 0.63$$

* Performing forward pass through neural n/w using input $[-0.32, -0.66]$ gives o/p 0.63 which is very different from actual value -0.54

Matrix form

$$h^1 = \begin{bmatrix} h_1^1 \\ h_2^1 \end{bmatrix}$$

$$w^2 = \begin{bmatrix} w_{11}^2 & w_{12}^2 \end{bmatrix} = \begin{bmatrix} 0.3 & 0.2 \end{bmatrix}$$

$$b^2 = [0.4]$$

$$h^2 = (w^2 h^1 + b^2) = \left(\begin{bmatrix} w_{11}^2 & w_{12}^2 \end{bmatrix}\begin{bmatrix} h_1^1 \\ h_2^1 \end{bmatrix} + b^2\right)$$

Loss Function

A loss function or cost function is a function that maps an event or values of one or more variables onto a real number.

In case of Regression, the most commonly used loss function is MSE/RSS.

In case of classification, the most commonly used loss function Cross Entropy / Log Loss.

In this example. predicted value is 0.63 & exp o/p is -0.54

$$\text{Loss}(L) = \frac{1}{2}(\text{actual} - \text{predicted})^2 = \frac{1}{2}(-0.54 - 0.63)^2 = 0.68$$

For classification :—

	Predicted	Actual
Lets assume	Prediction for class 1 = 0.20	1
	$P_{C2} = 0.70$	0
	$P_{C3} = 0.10$.	0

$$\text{Loss per Input} = -\sum y_j \log(P_j)$$

$$= -[1 * \log(0.20) + 0 * \log(0.70) + 0 * \log(0.10)]$$

$$= 0.69$$

$$CE = -\sum_{i=1}^{n} \sum_{j=1}^{k} y_{ij} \log(P_{ij})$$

How to minimise cost func^n :— tune parameters w & b This can be done using optimisation routine such as gradient descent.

Error Back Propagation

Training refers to the task of finding the optimal combination of weights & biases to minimize the total loss. This optimization is achieved using Gradient Descent algorithm.

$$Loss(L) = RSS = \sum (actual - h^2)^2$$
$$Loss(L) = f(w, b)$$

* To reduce the loss, all the weights have to be adjusted we initialise these weights with random values at the outset.

To adjust the weights Gradient Descent can be used

$$w_{kj}^{l} = w_{kj}^{l} - \eta \frac{\partial L}{\partial w_{kj}^{l}}$$ → derivative of Loss w.r.t. Loss func^n (Gradient of loss w.r.t weight or bias)

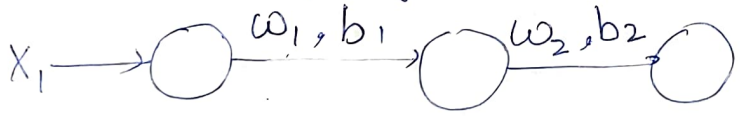
$$w_{11}^{1} = w_{11}^{1} - \eta \frac{\partial L}{\partial w_{11}^{1}} \qquad w_{21}^{1} = w_{21}^{1} - \eta \frac{\partial L}{\partial w_{21}^{1}} \qquad w_{11}^{2} = w_{11}^{2} - \eta \frac{\partial L}{\partial w_{11}^{2}}$$

$$w_{12}^{1} = w_{12}^{1} - \eta \frac{\partial L}{\partial w_{12}^{1}} \qquad w_{22}^{1} = w_{22}^{1} - \eta \frac{\partial L}{\partial w_{22}^{1}} \qquad w_{12}^{2} = w_{12}^{2} - \eta \frac{\partial L}{\partial w_{12}^{2}}$$

$$b_1^{1} = b_1^{1} - \eta \frac{\partial L}{\partial b_1^{1}} \qquad b_2^{1} = b_2^{1} - \eta \frac{\partial L}{\partial b_2^{1}} \qquad b_1^{2} = b_1^{2} - \eta \frac{\partial L}{\partial b_1^{2}}$$

To compute these gradients we use algorithm called Backpropagation.

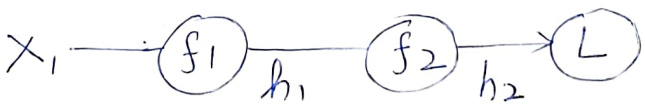
We will simplify the neural n/w as



(one neuron in hidden layer, one neuron in O/P layer)

Lets assume the loss func^n that we are using is MSE

$$Loss = \frac{1}{2}(y - h_2)^2$$



$$h_i = \tanh(z_i)$$
$$z_i = w_i h_{i-1} + b_i$$

Activation func^n used is tanh

* If we change the weight & ~~loss~~ bias then Loss func^n is also changed.

How we will change weight term w_2 :-

First compute gradient of Loss w.r.t. w_2

$$\frac{\partial L}{\partial w_2}$$

changing weight 2 (w_2) will change Z_2 which in turn will change the value of h_2, which in turn will change the value of Loss (L). So there is an indirect relationship between w_2 and Loss.

So change in L w.r.t. w_2 can be represented as a product of change in L w.r.t. activation h_2 & change in Z_2 due to change in w_2

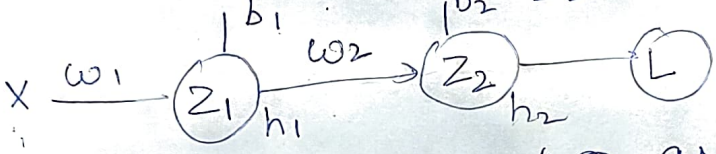
$$\frac{\partial L}{\partial w_2} \Leftarrow \frac{\partial L}{\partial h_2} \frac{\partial h_2}{\partial Z_2} \frac{\partial Z_2}{\partial w_2} \quad \text{— eq}^n \text{ (1)}$$

↑ Gradient of Loss with w_2

↑ Gradient of L with h_2

↑ Gradient of h_2 with Z_2

Gradient of Z_2 with w_2



we can simplify the eq^n (1) as

$$\frac{\partial h_2}{\partial w_2} = \frac{\partial h_2}{\partial Z_2} \frac{\partial Z_2}{\partial w_2}$$

$$\therefore \frac{\partial L}{\partial w_2} \Leftarrow \frac{\partial L}{\partial h_2} \frac{\partial h_2}{\partial w_2} \quad \text{— eq}^n \text{ (2)}$$

Changing the order for mathematical simplicity we can rewrite eq^n (2) as

$$\frac{\partial L}{\partial w_2} \Leftarrow \frac{\partial h_2}{\partial w_2} \frac{\partial L}{\partial h_2}$$

$$L = \frac{1}{2}(y - h_2)^2 \rightarrow \frac{\partial L}{\partial h_2} = \frac{\partial}{\partial h_2} \frac{1}{2}(y - h_2)^2 = -(y - h_2) \quad \text{— (1)}$$

$$h_2 = \tanh(Z_2) \Rightarrow \frac{\partial h_2}{\partial Z_2} = 1 - \tanh^2(Z_2) = 1 - (h_2)^2 \quad \text{— (2)}$$

$$Z_2 = w_2 h_1 + b_1 \Rightarrow \frac{\partial Z_2}{\partial w_2} = h_1 \quad \text{— (3)}$$

Substituting the values

$$\frac{\partial L}{\partial w_2} \Leftarrow [-(y-h_2)] * [-(h_2)^2] * [h_1]$$

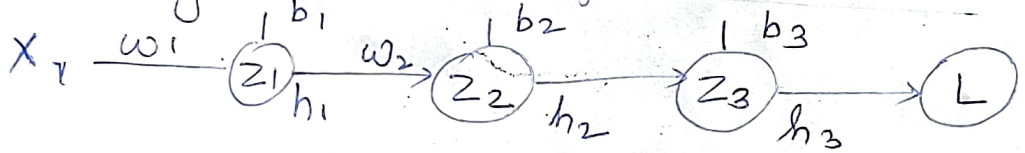
$$w_2 = w_2 - \eta \frac{\partial L}{\partial w_2}$$

$$w_1 = w_1 - \eta \frac{\partial L}{\partial w_1}$$

Same to be computed for w_1

$$\frac{\partial L}{\partial w_1} \Leftarrow \frac{\partial h_1}{\partial w_1} \frac{\partial L}{\partial h_1} \Rightarrow \frac{\partial L}{\partial h_1} = \frac{\partial h_2}{\partial h_1} \frac{\partial L}{\partial h_2}$$

Extending this model for 3 neurons



$$w_1 = w_1 - \eta \frac{\partial L}{\partial w_1} \qquad w_2 = w_2 - \eta \frac{\partial L}{\partial w_2} \qquad w_3 = w_3 - \eta \frac{\partial L}{\partial w_3}$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial h_1}{\partial w_1} \frac{\partial L}{\partial h_1} \qquad \frac{\partial L}{\partial w_2} = \frac{\partial h_2}{\partial w_2} \frac{\partial L}{\partial h_2} \qquad \frac{\partial L}{\partial w_3} = \frac{\partial h_3}{\partial w_3} \frac{\partial L}{\partial h_3}$$

$$\frac{\partial L}{\partial w_3} \frac{\partial L}{\partial h_3} = \frac{\partial}{\partial h_3} \frac{1}{2}(y-h_3)^2 = -(y-h_3)$$

$$\frac{\partial L}{\partial h_2} = \frac{\partial h_3}{\partial h_2} \frac{\partial L}{\partial h_3}$$

$$\frac{\partial L}{\partial h_1} = \frac{\partial h_2}{\partial h_1} \frac{\partial L}{\partial h_2}$$

$$\frac{\partial L}{\partial h_3} = -(y-h_3)$$

$$\frac{\partial L}{\partial h_1} = \frac{\partial h_2}{\partial h_1} \frac{\partial L}{\partial h_2} \qquad \frac{\partial L}{\partial h_2} = \frac{\partial h_3}{\partial h_2} \frac{\partial L}{\partial h_3} \qquad \frac{\partial L}{\partial h_3} = -(y-h_3)$$

$$\frac{\partial L}{\partial h_1} = \frac{\partial h_2}{\partial h_1} \frac{\partial h_3}{\partial h_2} \frac{\partial L}{\partial h_3}$$

$$\frac{\partial L}{\partial h_1} = \frac{\partial h_2}{\partial h_1} \frac{\partial h_3}{\partial h_2} [-(y-h_3)]$$

* we are moving in backward direction

Pseudocode

1. Initialize with the input

Fowaad propagation

2. For each layer compute the cumulative input and apply the activation function on the cumulative input of each neuron of each layer to get the output.
3. For classification, get the probabilities of the observation belonging to a class & for regression compute the numeric output
4. Assess the performance of neural through loss function, for example cross entropy loss func^n for classification & RMSE for regression.

Back propagation

5. From the last layer to first layer, for each layer compute gradient of the loss func^n w.r.t. the weights at each layer & all the intermediate gradients
6. Once all the gradients of the loss w.r.t. the weights (and biases) are obtained, use an optimisation technique like gradient descent to update the values of the weights & biases.
7. Repeat this process until the model gives acceptable predictions.

Regularization

→ NN are large, complex models & have tendency to overfit the training data. Regularization is a technique to address this problem.

Objective func^n = Loss function (Error term) + Regularization term

↑ Reducing this term will have bias ↓ & variance ↑

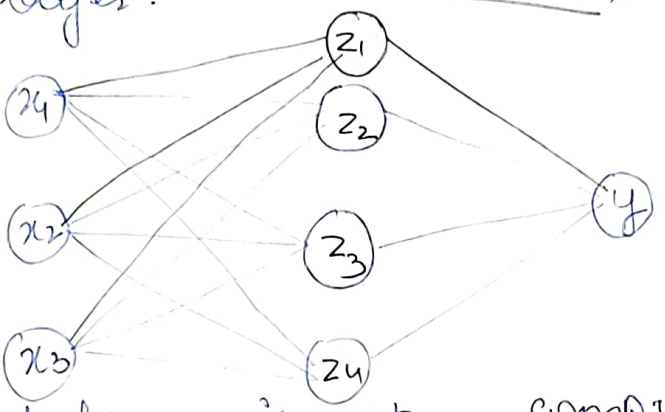
↑ bias ↑ variance ↓

Objective func^n = $L(F(x_i), \theta) + \lambda f(\theta)$

θ → model parameter

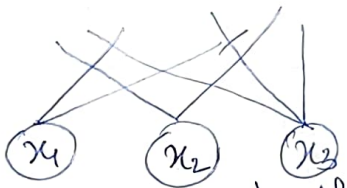
Radial Basis Function

→ used in ANN
→ has only one hidden nodes, one input layer & an output layer.



The input layer is not a computation layer, it just receives the input data and feeds it into the special hidden layer of the RBF network. The computation that is happened inside the hidden layer is very different from most neural n/w. The output layer performs the prediction task such as classification or regression

Input layer



The number of neurons in the input layer should be equal to the dimensionality of the data. The input neurons are fully connected to the hidden neurons & feed their input forward.

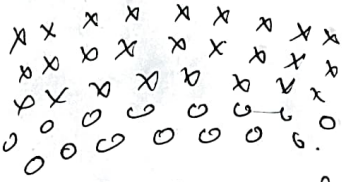
Hidden layer

The hidden layer takes the input in which the pattern might not be linearly separable & transform it into a new space that is more linearly separable. The hidden layer has higher dimensionality than the input layer because the pattern that is not linearly separable often needs to be transformed into higher-dimensional space to be more linearly separable. This is based on Cover's theorem on separability of patterns which states that a pattern that is transformed into a higher

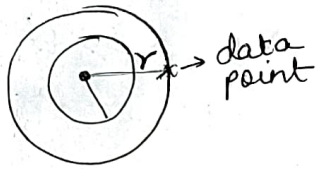
dimensional space with non linear transformation is more likely to be linearly separable, therefore the number of neurons in the hidden layer should be greater than the number of the input neuron.

How RBF does this?

- Consider one center randomly
- draw cocentric circles



* Data is not linearly separable.



we use Gaussian function

$$\phi(r) = \exp\left[\frac{-r^2}{2\sigma^2}\right]$$

$\sigma > 0$, constant