Types of attributes / features / dimensions / variables.

→ we want to transform those features (numerical) have non-numeric values in the way so we may able to perform ML on them.

Q what is attribute?

A data field that representing a characteristic or feature of a data object.

E.g customer-ID, name etc

Types of attributes w.r.t to data they contained

- Nominal :- categories, states (labels, Qualities) or names of things.
  Hair-color, marital status, occupation, ID numbers, zip codes
  (Order of value is not important)
- Binary : Nominal attributes with only 2 states (0 and 1)
  (i) Symmetric (ii) Asymmetric
  (less important outcome / Most important outcome)
- Ordinal :- values have meaningful order (ranking) but magnitude of difference b/t them is not known/quantifiable. e.g size, grades, army Ranks
  (Order of value is important)
- Numeric : Quantitative
  - Interval :- No true-zero point exist / starting point is not 0 (Base value is not 0)
    0°C / F°, calender dates.
  - Ratio
    True-zero point exist / starting point 0 / (Base value is 0)
    Kelvin

Discrete attributes → Represented by integers

Continous attributes → Represented by real values (floating values)

→ we assign Id to nominal attributes.

Types of attributes / features / dimensions / variables.

→ we want to transform features (those) have non-numeric values in the (numerical) way so we may able to perform ML on them.

Q what is attribute?

A data field that representing a characteristic or feature of a data object.

E.g customer-ID, name etc

Types of attributes W.r.t to data they contained

- Nominal :- categories, states (labels, Qualities) or names of things.
  - Order of value is not important
  - Hair-color, marital status, occupation, ID numbers, ZIP codes
  - Binary : Nominal attributes with only 2 states (0 and 1)
    - (i) Symmetric (ii) Asymmetric
- Ordinal :- values have meaningful order (ranking) but magnitude between them is not known / quantifiable e.g size, grades, army Ranks
  - Order of value is important
  - Less important outcome, Most important outcome
- Numeric : Quantitative
  - Interval :- No true-zero point exist / starting point is not 0 (Base value is not 0)
    - °C / F°, calender dates.
  - Ratio
    - True-zero point exist / starting point 0 / Kelvin (Base value is 0)

Discrete attributes → Represented by integers

Continous attributes → Represented by real values (floating values)

→ we assign Id to nominal attributes.

Similarity and Dissimilarity between attributes.

- Proximity:- refers to a similarity or dissimilarity
- Similarity are

Numerical measure that tells how alike two objects
value is higher when objects are more alike
values fall in range [0, 1]
0 means objects not matched/similar at all.
1 means objects exactly matched/similar

Dissimilarity

Numerical measure that tells how different two objects are.
value is higher when objects are more different.
Range for dissimilarity [0, varies
0 means objects exactly matched/similar or less dissimilar
upper limit varies.

Data matrix.

n = no. of example point in dataset.
p = no of features/dimension/attributes/variables of every example point.

$$\begin{bmatrix} x_{11} \cdots x_{1f} \cdots x_{1p} \\ \cdots \cdots \cdots \cdots \\ x_{i1} \cdots x_{if} \cdots x_{ip} \\ \cdots \cdots \cdots \cdots \\ x_{n1} \cdots x_{nf} \cdots x_{np} \end{bmatrix}_{n \times p} \quad \text{e.g} \quad \begin{bmatrix} 1 \cdots 5 \cdots 10 \\ 2 \\ 50 \\ \vdots \\ 100 \end{bmatrix}_{100 \times 10}$$

we make dissimilarity matrix on the basis of n.

Dissimilarity Matrix.

- n example points but registers only the distance.
- A triangular matrix of size n x n.

→ Distance formula is dissimilarity measure.
→ Example point = Data point = Data object.

If n=4 then dissimilarity matrix is a triangular matrix of size nxn = 4x4
n = E1, E2, E3, E4 i,j = 1,2,3,4.

	E1	E2	E3	E4
E1	0			
E2		0		
E3			0	
E4				0

nxn

Proximity measure for nominal attributes.
- Attributes can take 2 or more states
(Generalization of Binary attributes)
Methods:- Simple matching.
(1) First to calculate total no. of Nominal attributes in dataset
P = Total no. of nominal attributes in dataset.
m = No. of matches
then

$$d(i,j) = \frac{P-m}{P} \qquad S(i,j) = \frac{m}{P}$$

Proximity measure for Binary attributes
contingency table for binary attribute

		Object j: 1	Object j: 0	Sum
Object i:	1	q	r	q+r
	0	s	t	s+t
	Sum	q+s	r+t	

Total matches = q+t
Total mismatches = r+s
Total = q+r+s+t.

For Symmetric Binary attributes
$d(i,j) = \frac{s+r}{q+r+s+t}$
$S(i,j) = \frac{q+t}{q+r+s+t}$

For Asymmetric Binary attributes.
$d(i,j) = \frac{r+s}{q+r+s}$
$S(i,j) = \frac{q}{q+r+s}$

Jaccard coefficient (similarity measure)

Standardization/Normalization of Numeric Data.

① Min-Max Normalization: to $[new\_min_A, new\_max_A]$

$$\hat{V} = \frac{V - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A$$

Here:

V represents numeric feature value

$\hat{V}$ represents transformed/normalized numeric feature value.

min_A: minimum value of feature before normalization

max_A: maximum value of feature before normalization

new_min_A: new minimum value of feature that you want to set after normalization

new_max_A: new maximum value of feature that you want to set after normalization

Example let feature (income) range 12000 to 98000 normalized to [0.0, 1.0]. Then 73600 is mapped to

$$\frac{73600 - 12000}{98000 - 12000}(1.0 - 0.0) + 0.0 = 0.716$$

② z-score normalization (μ: mean, σ: standard deviation)

$$\hat{V} = \frac{V - \mu_A}{\sigma_A}$$

Here

μ_A: mean of attribute/feature

σ_A: standard deviation of attribute/feature

Example let $\mu = 54000$, $\sigma = 16000$ then 73600 is mapped to

$$= \frac{73600 - 54000}{16000} = 1.225$$

③ Normalization by decimal scaling

$\hat{V} = V/10^j$ where j is the smallest integer such that $max(|\hat{V}|) < 1$

Special cases of Minkowski Distance. $d(i,j) = \left(\sum_{f=1}^{p} |x_{if} - x_{jf}|^h\right)^{1/h}$.

① $h=1$: Manhattan (city block, L_1 norm) distance.

E.g. the Hamming distance: the number of bits that are different between two binary vectors.

$$d(i,j) = \left[\sum_{f=1}^{p} |x_{if} - x_{jf}|^h\right]^{1/h} = \sum_{f=1}^{p} |x_{if} - x_{jf}|$$
$$= |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \cdots + |x_{ip} - x_{jp}|$$

② $h=2$: Euclidean (L_2 norm) distance.

$$d(i,j) = \left(\sum_{f=1}^{p} |x_{if} - x_{jf}|^h\right)^{1/h} = \left(\sum_{f=1}^{p} |x_{if} - x_{jf}|^2\right)^{1/2}$$
$$= \sqrt{|x_{i1} - x_{j1}|^2 + |x_{i2} - x_{j2}|^2 + \cdots + |x_{ip} - x_{jp}|^2}$$

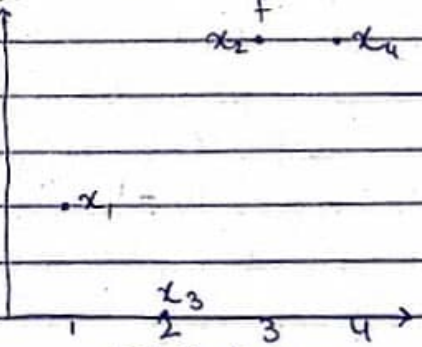
③ $h \to \infty$: Supremum (L_{max} norm, L_∞ norm) distance.

This is the maximum difference between any component attribute of the vectors.

$$d(i,j) = \lim_{h \to \infty} \left(\sum_{f=1}^{p} |x_{if} - x_{jf}|^h\right)^{1/h} = \max_{f}^{p} |x_{if} - x_{jf}|$$

Example

Points	attribute 1	attribute 2
x_1	1	2
x_2	3	5
x_3	2	0
x_4	4	5



Dissimilarity Matrices.

Manhattan (L_1)

	x_1	x_2	x_3	x_4
x_1	0			
x_2	5	0		
x_3	3	6	5	0
x_4	6	1	7	5

Euclidean (L_2)

	x_1	x_2	x_3	x_4
x_1	0			
x_2	3.61	0		
x_3	2	5.1	0	
x_4	4.24	1	5.39	0

Supremum (L_∞)

	x_1	x_2	x_3	x_4
x_1	0			
x_2	3	0		
x_3	2	5	0	
x_4	3	1	5	0

Ordinal variable/attributes.

- An ordinal variable can be discrete or continous
- order is important e.g rank
- can be treated like interval-scaled
  - Assign id's to all ranks or to all possible values of ordinal feature
  - for example universities teachers designations.

designation	lecturer	Assistant Lecturer	Associat Prof	Professor
id's	1	2	3	4

- $r_{ip} \in \{1 \dots M_p\}$.
  - r_{ip} represents corresponding id or rank value to i^{th} example point in th f^{th} feature (ordinal feature)
  - 1 represents lowest rank
  - M represents Maximum/highest rank.
- Replace x_{if} by r_{if}
  - x_{if} represents i^{th} example point in th f^{th} feature.
- After replacing map f^{th} feature onto [0, 1] by Min-Max Normalization

$$z_{if} = (r_{if} - 1)/(M_f - 1)$$

- compute the dissimilarity using methods for interval scaled like .. Euclidean Distance method

Attributes of Mixed Type

- A database may contain all attribute types (Nominal, Binary, Symmetric Asymmetric binary, numeric, ordinal). then compute distance/dissimilarity matrices for all types
- one may use a weighted formula to combine their effects.

$$d(i,j) = \sum_{f=1}^{p} \delta_{ij}^{(f)} d_{ij}^{(f)} / \sum_{f=1}^{p} \delta_{ij}^{(f)}$$

→ f is binary or nominal $d_{ij}^{(f)} = 0$ if $x_{if} = x_{jf}$ or $d_{ij} = 1$ otherwise

→ f is numeric use normalized distance $d_{ij}^{(f)} = \frac{|x_{if} - x_{jf}|}{\max_h x_{hf} - \min_h x_{hf}}$

→ f is ordinal

compute ranks r_{if} and z_{if} as interval scaled $z_{if} = \frac{r_{if} - 1}{M_f - 1}$

Simultaneous equation is not scaleable but GD is.

Gradient Descent

- Gradient Descent is just like Agile Methodology
  - Agile Methodology (Iterative approach)
    1. Build something quickly (Prototype).
    2. Get it out there (deploye or Test)
    3. Get some feedback (shared with customer)
    4. Make changes depending upon the feedback
    5. Repeat.
- Similarly agile methodology we use iterative approach in gradient descent first we quickly build model and then evaluate it and compare results with actuals results and repeat this process.
- Example.
- lets have some function say $j(\theta)$. ($j(\theta)$: may be one value; θ: may be vector.) θ may be one parameter or list of parameters. $\theta = [\theta_0, \theta_1, \cdots \theta_n]$
- objective function
  Either we maximize or minimize the function ($j(\theta)$) by finding specific values of θ. $\max_{\theta} j(\theta)$ or $\min_{\theta} j(\theta)$.
- lets we want to find such values of θ for which $j(\theta)$ return minimum value. So to achieve this objective we use gradient descent
- Working of GDA — search for weights: Random initialization technique.
  - Randomly initialize θ's. { usually range is -0.5 to $+0.5$ (i.e. -1 to $+1$ around 0) }
  - Keep changing θ's to reduce/minimize $j(\theta)$ until we hopefully end up at a minimum value of $j(\theta)$ by using weight updated equations.
    - repeat until convergence.

$$\theta_i := \theta_i - \alpha \frac{\partial}{\partial \theta_i} j(\theta) \rightarrow \text{weight update equation}$$

convex concave

Weight update equation for minimization

$$\theta_{i+1} = \theta_i - \alpha \frac{\partial}{\partial \theta_i} j(\theta)$$

Weight update equation for maximization.

$$\theta_{i+1} = \theta_i + \alpha \frac{\partial}{\partial \theta_i} j(\theta)$$

α is called eta
- step size.
- learning rate.

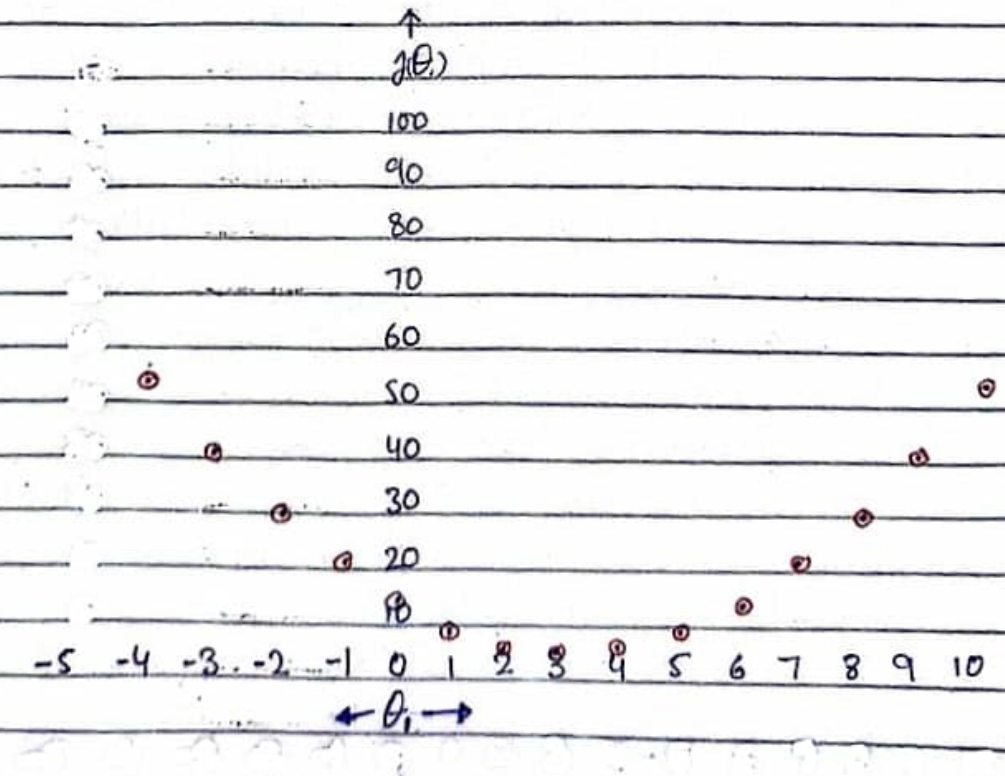
operators	In Programming languages	In Mathematics
Assignment	=	:=
Equal	==	=

→ Derivative return rate of change or slope of function.

Day sun

lets there is minimization convex problem and function is $j(\theta_i) = (\theta_i - 3)^2 + 5$.

θ_i	$j(\theta_i)$
-4	54
-3	41
-2	30
-1	21
0	14
1	9
2	6
3	5
4	6
5	9
6	14
7	21
8	30
9	41
10	54



→ No. of weight update equation = No. of ~~unknown parameter~~ weights in function.
So by using GD for minimization

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} j(\theta_1)$$

Q:-

$$\frac{\partial}{\partial \theta_1} j(\theta_1) = \frac{\partial}{\partial \theta_1} (\theta_1 - 3)^2 + 5 = 2(\theta_1 - 3)^1 \cdot \frac{\partial}{\partial \theta_1}(\theta_1 - 3) + \frac{\partial}{\partial \theta_1}(5) = 2(\theta_1 - 3)$$

Let $\alpha = 0.1$ so

$$\theta_1 := \theta_1 - 0.1 \times 2(\theta_1 - 3).$$

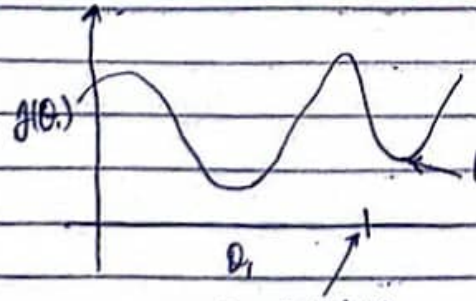
Q:- Randomely initialize θ_1 and sum two iteration of GD lets $\theta_1 = 10$ and $\alpha = 0.1$ and objective function $\min_{\theta_1} j(\theta_1)$.

$$\theta_1 = 10 - 0.1 \times 2(10 - 3)$$
$$= 8.6$$
$$= 8.6 \times 0.1 \times 2(8.6 - 3)$$
$$= 7.48$$

lets $\theta_1 = -5$ and $\alpha = 0.1$

$$\theta_1 = -5 - 0.1 \times 2(-5 - 3)$$
$$= -3.4$$
$$\theta_1 = -3.4 \times 0.1 \times 2(-3.4 - 3)$$
$$= -2.12$$

So you see θ_1 value approaches 3 and after many iteration θ_1 value converged to 3 and $\frac{\partial}{\partial \theta_1} j(\theta_1)$ this factor return 0 because slope at $\theta_1 = 3$ is zero.

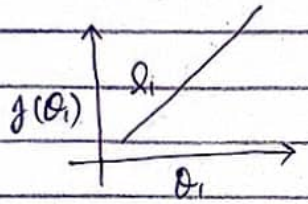


$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} j(\theta_1).$$

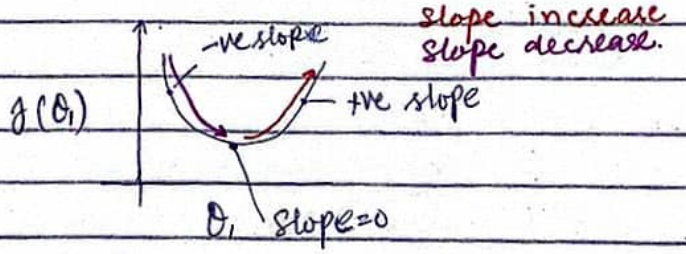
If you have $j(\theta)$ as plot as you reach at local optima and want to update θ_1 you can't do that

Impact of learning rate in GD (usually in range (0-1))

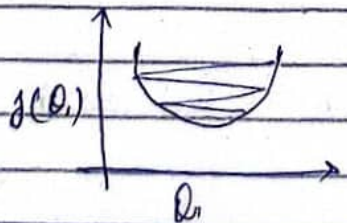
- If α is too small, GD can be slow. More time and iterations are required to achieve goal.
- As slope of line is same at every point and by taking too small value of α GD slowly approached its goal (minimum value of $J(\theta)$) by taking more time and iterations. but approached its goal with same speed



- As slope of curve is not same at every point and by taking too small value of α GD slowly approached its goal by taking more time and iterations but initially GD try to approache its goal fastly but become slow over the period of time



- If α is too large, GD can overshoot the minimum. It may fail to converge, or even diverge.



- if $\alpha = 0$ weights will not update
- if $\alpha < 0$ flip the direction of update essentially moving away from the optimal solution.

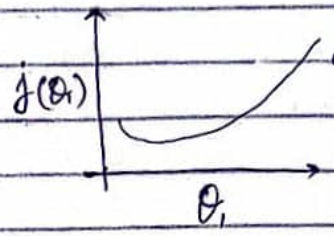
→ α value depend on function. → α is hyperparameter.

eta value should be large.

eta value should be small.

- GD can converge to a local minimum, even with the fixed learning rate (α)

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$



As we approach a local minimum, GD will take automatically smaller steps. due to the impact of slope. so no need to decrease α over time. Therefore we initially take small +ve value of eta and gradually increase it over the period of time.

- There is some cost function lets $J(\theta_0, \theta_1)$
- Objective function is to minimize cost function $J(\theta_0, \theta_1)$

$$\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$$

- we minimize $J(\theta_0, \theta_1)$ by GD.

For using GD to minimize $J(\theta_0, \theta_1)$ following steps should be taken.

① Find weight update equation for every weight.
- partial derivative of cost function $J(\theta_0, \theta_1)$ w.r.t weights

eg for θ_0, θ_1

$$\theta_0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1).$$

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1).$$

② Randomly initialize weights.

③ Repeat until weights not converged, by using weight update equation.

$$\theta_i := \theta_i - \alpha \frac{\partial}{\partial \theta_i} J(\theta_i)$$

(3.1) There are two approaches to stop iteration
① check desired result (MSE is 0 or minimum) is achieved.
② check unknown parameters are converged or not.

(4) Simultaneously update weights.

Correct way

$temp_0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} j(\theta_0, \theta_1)$

$temp_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} j(\theta_0, \theta_1)$

$\theta_0 := temp_0$

$\theta_1 := temp_1$

Incorrect way

$temp_0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} j(\theta_0, \theta_1)$

$\theta_0 := temp_0$

$temp_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} j(\theta_0, \theta_1)$

$\theta_1 = temp_1$

Gradient Descent for linear Regression.

Hypothesis function: tell how to predict values.

Cost function: tell which unknown parameters are best.

Objective function: tell either minimize or maximize the cost function

Approach (GD, Simultaneous equation ..): To find unknown parameters

Linear Regression.

Hypothesis function $y'^{(i)} = h_\theta(x^i) = \theta_0 + \theta_1 x^i$

Hypothesis function for multiple features $y'^i = h_\theta(x^i) = \theta_0 + \theta_1 x_1^i + \theta_2 x_2^i + \cdots$

Cost function = MSE $= j(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (y'^i - y^i)^2$

Objective function $= \min_{\theta_0, \theta_1} j(\theta_0, \theta_1)$

Approach = Gradient Descent.

① weight update equation for each weight.

$\theta_0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} j(\theta_0, \theta_1)$; $\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} j(\theta_0, \theta_1)$

$\frac{\partial}{\partial \theta_0} j(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_0} \frac{1}{2m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x^i - y^i)^2$

$\frac{\partial}{\partial \theta_1} j(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_1} \frac{1}{2m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x^i - y^i)^2$

These already solved in linear regression lecture.

$$\frac{\partial}{\partial \theta_0} j(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x^i - y^i) = \frac{1}{m} \sum_{i=1}^{m} (y'^i - y^i)$$

$$\frac{\partial}{\partial \theta_1} j(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x^i - y^i) x^i = \frac{1}{m} \sum_{i=1}^{m} (y'^i - y^i) x^i$$

so.

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (y'^i - y^i) \quad \text{and} \quad \theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (y'^i - y^i) x^i$$

$\sum_{i=1}^{m}$ represent all example points that why this is example of Batch Gradient Descent.

① Randomly initialize weights

③ Repeat until weights not converged by using desired weight update equations

④ Simultaneously update weights.

Dry run on training data.

lets $\alpha = 0.1$

		initialize. Randomly	1st iteration Updated weights	2nd Iteration Updated weights
		$\theta_0 = 3, \theta_1 = 7$	$\theta_0 = 2.6, \theta_1 = 5.4$	$\theta_0 = 2.72, \theta_1 = 5.68$
X = inputs	Y = output	Y_1'	Y_2'	Y_3'
1	10	10	8	8.38
2	15	17	13.4	14.06
3	20	24	18.8	19.74
4	25	31	24.2	25.42
5	30	38	29.6	31.1

Assignment.

Impliment Linear Regression by using GD. for $\alpha = 0.1, 0.001, 0.01$ and range for random initialization of weight is [0 to 1].

Q: $E(w) = w_1^2 + w_2^2$ run two iteration of GDA.
assum eta = $\eta = \alpha = 0.1$, $w_1 = 8$ $w_2 = 10$.

sol for minimization.

$$w_1 := w_1 - \alpha \frac{\partial}{\partial w_1} E(w) \qquad w_2 := w_2 - \alpha \frac{\partial}{\partial w_2} E(w)$$

$$w_1 := w_1 - \alpha 2w_1 \qquad w_2 := w_2 - \alpha 2w_2$$

$= 8 - 0.1 \times 2 \times 8$ $\qquad = 10 - 0.1 \times 2 \times 10$

$= 6.4$ (after 1st iteration). $\qquad = 8$ (after 1st iteration)

$= 6.4 - 0.1 \times 2 \times 6.4$ $\qquad = 8 - 0.1 \times 2 \times 8$.

$= 5.12$ (after 2nd iteration) $\qquad = 6.4$ (after 2nd iteration)

Derivation of weight update equation by chain rule in linear Regression

Hypothesis function $h_\theta(x^i) = y'^i = \theta_0 + \theta_1 x^i$.

cost function = MSE = $j(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (y'^i - y^i)^2$

Objective function $\min_{\theta_0, \theta_1} j(\theta_0, \theta_1)$

chain Rule $\frac{dy}{dx} = \frac{dy}{dz} \cdot \frac{dz}{dx}$.

Required weight update equations

$\theta_0 := \theta_0 - \eta \Delta\theta_0$ and $\theta_1 := \theta_1 - \eta \Delta\theta_1$

$\Delta\theta_0 = \frac{\partial}{\partial \theta_0} j(\theta_0, \theta_1)$ $\qquad \Delta\theta_1 = \frac{\partial}{\partial \theta_1} j(\theta_0, \theta_1)$

$= \frac{\partial j(\theta_0, \theta_1)}{\partial y'} \cdot \frac{\partial y'}{\partial \theta_0}$ $\qquad = \frac{\partial j(\theta_0, \theta_1)}{\partial y'} \cdot \frac{\partial y'}{\partial \theta_1}$

$= \frac{\alpha}{2m} \sum_{i=1}^{m} (y'^i - y^i) \cdot 1$ $\qquad = \frac{\alpha}{2m} \sum_{i=1}^{m} (y'^i - y^i) \cdot x^i$

so $\theta_0 := \theta_0 - \eta \frac{1}{m} \sum_{i=1}^{m} (y'^i - y^i)$ $\qquad \theta_1 := \theta_1 - \eta \frac{1}{m} \sum_{i=1}^{m} (y'^i - y^i) \cdot x^i$

Simultaneously updates weights.

Reuse these factors values $\frac{1}{m}\sum_{i=1}^{m}(y'^i - y^i)$, $\frac{1}{m}\sum_{i=1}^{m}(y'^i - y^i)\cdot x^i$

$$temp_0 = \frac{1}{m}\sum_{i=1}^{m}(y'^i - y^i) \qquad temp_1 = \frac{1}{m}\sum_{i=1}^{m}(y'^i - y^i)\cdot x^i$$

Now updates weights.

$\theta_0 := \theta_0 - \gamma\, temp_0$ and $\theta_1 := \theta_1 - \gamma \cdot temp_1$.

Final Exam

- Extensive discussion on hyperparameters, model over-fitting, model under-fitting, cross validation, information leaking.
- Question on GD
- Pre-mid not included
- Types of attributes topic included in final.
- Softmax related question.

Term project assignment

- 4 times effort required than
- always split data into training validation and testing
- validation data set for tuning hyperparameter.
- select your topic
- Extensive use of library
- data set and libraries required for project
- Not used already learned algorithm
- Kaggle, Hugging face.

:Classification problems:- ① In classification problems output is categorical, countable or discrete. ② we classify data into pre-defined classes.

e.g binary classification problems, multiclassification problems multi-level classification problems.

Linear classification problem

- classes are linearly separable
- classes can be separated by linear decision boundary

Non linear classification problems:

- classes are not linearly separable.
- classes can be separated by non-linear decision boundary.

→ we can relate AND gate logic and OR gate logic to a linear classification problem with two input features and binary outputs.

x_1	x_2	AND=y
0	0	0
0	1	0
1	0	0
1	1	1

$y' = \theta_0 + \theta_1 x_1 + \theta_2 x_2$

→ Range of $\theta_0, \theta_1, \theta_2, x_1$ and x_2 is $(-\infty, +\infty)$ but if we fix range of x_1 & x_2 [0, 1] it not affect θ_0, θ_1 and θ_2 range. $(-\infty, +\infty)$

Threshold :- the level or point at which you start to experiment something.

Defining Threshold: Threshold required for classification problem but not for regression problem because in case of regression output is continous value while in case of classification output is discrete and we required threshold to classify the data.

lets for above example threshold is 3.

If $y' \geq 3$
class 1
else
class 0.

Q1

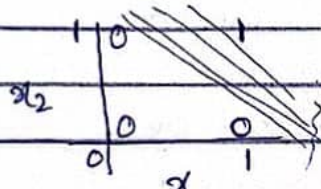
So for what values of θ_0, θ_1 and θ_2 following equation valid ... $y' = \theta_0 + \theta_1 x_1 + \theta_2 x_2$ and threshold is 3.

Sol

x_1	x_2	AND = y	$y' = \theta_0 + \theta_1 x_1 + \theta_2 x_2$
0	0	0	$\theta_0 < 3$
0	1	0	$\theta_0 + \theta_2 < 3$
1	0	0	$\theta_0 + \theta_1 < 3$
1	1	1	$\theta_0 + \theta_1 + \theta_2 \geq 3$

if $y' \geq 3$ class 1 else class 0.

So these are many solutions as we see when we plot AND gate logic



many solution.

①

θ_0	1	0
θ_1	1	1
θ_2	1	2

②

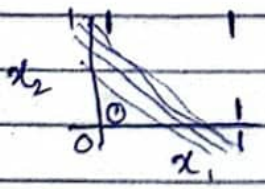
OR Gate

Lets Threshold is 7.

x_1	x_2	OR = y	$y' = \theta_0 + \theta_1 x_1 + \theta_2 x_2$
0	0	0	$\theta_0 < 7$
0	1	1	$\theta_0 + \theta_2 \geq 7$
1	0	1	$\theta_0 + \theta_1 \geq 7$
1	1	1	$\theta_0 + \theta_1 + \theta_2 \geq 7$

if $y' \geq 7$ class 1 else class 0.

So these are many solutions as we plot OR gate logic



⑤

θ_0	3	5
θ_1	4	2
θ_2	5	2

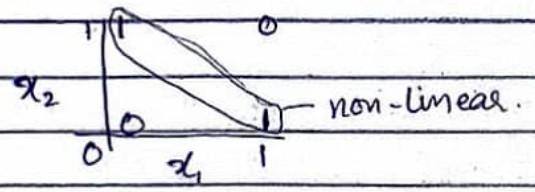
②

we can't relate XOR gate logic to linear classification problem because classes can't be separated by linear decision boundary rather we can relate it with non linear classification problem

x_1	x_2	y = XOR	$y' = \theta_0 + \theta_1 x_1 + \theta_2 x_2$
0	0	0	$\theta_0 < 5$
0	1	1	$\theta_0 + \theta_2 \geq 5$
1	0	1	$\theta_0 + \theta_1 \geq 5$
1	1	0	$\theta_0 + \theta_1 + \theta_2 < 5$.

Lets Threshold is 5.
If $y' \geq 5$ class 1
else class 0.

so there is no linear solution exist as we see when plot XOR gate logic we can't separate classes by single line (linear) but can be separated by oval (non-linear)



①

θ_0	no solution exist.
θ_1	
θ_2	

②

→ If there are more than 2 classes (multiclass problem)
① 1 class vs all classes approach is used for classification
② Define different Thresholds.

exponential function = $f(x) = a^x$.
$a > 0$, $a \neq 1$ and x is any real no.
Natural exponential function = $f(x) = e^x$

Sigmoid Derivation	$\frac{d}{dx} a^x = a^x \cdot \ln a$	$\frac{d}{dx} a^{\sqrt{x}} = a^{\sqrt{x}} \cdot \ln a \cdot \frac{d}{dx}\sqrt{x}$
$\sigma(x) = \frac{1}{1+e^{-x}}$	$\frac{d}{dx} e^x = e^x$	$\frac{d}{dx} e^{x^2+1} = e^{x^2+1} \cdot \frac{d}{dx}(x^2+1)$.

$\sigma'(x) = \frac{d}{dx}\sigma(x) = \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right)$

$= \frac{d}{dx}(1+e^{-x})^{-1} = -1(1+e^{-x})^{-2} \frac{d}{dx}(1+e^{-x}) = \frac{-1}{(1+e^{-x})^2}\left(\frac{d}{dx}(1) + \frac{d(e^{-x})}{dx}\right)$

$= \frac{-1}{(1+e^{-x})^2}\left(0 + e^{-x} \cdot \frac{d(-x)}{dx}\right) = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1 \cdot e^{-x}}{(1+e^{-x})(1+e^{-x})}$

$= \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}} = \sigma(x) \cdot \frac{1+e^{-x}-1}{1+e^{-x}} = \sigma(x) \cdot \left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right)$

$= \boxed{\sigma(x)(1-\sigma(x))}$

Binary cross Entropy / log loss Derivation

- For two class problem – cost / loss function for binary classification problems.
- cost / loss function for logistic regression

Cost / lost function: $J(\theta) = \frac{-1}{m} \sum_{i=1}^{m} \left[y^i \log(y'^i) + (1-y^i)\log(1-y'^i)\right]$

Objective function: $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$ or $\min_{\theta} J(\theta)$.

Hypothesis function: $h_\theta(x^i) = y'^i = \sigma(\theta_0 + \theta_1 x^i)$ $\quad \theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 \ldots$

Lets derive weight update equations

$\theta_j = \theta_0, \theta_1, \theta_2 \ldots \theta_j$.

$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{\partial}{\partial \theta_j}\left(\frac{-1}{m}\sum_{i=1}^{m}\left[y^i \log(y'^i) + (1-y^i)\log(1-y'^i)\right]\right)$

$= \frac{-1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial \theta_j}\left[y^i \log(y'^i) + (1-y^i)\log(1-y'^i)\right]$

$= \frac{-1}{m}\sum_{i=1}^{m}\left[y^i \frac{\partial}{\partial \theta_j}\log(y'^i) + (1-y^i)\frac{\partial}{\partial \theta_j}(1-y'^i)\right]$

$\because \frac{d}{dx}\log x = \frac{1}{x}\cdot\frac{d}{dx}(x) = \frac{1}{x}$

$\because \frac{d}{dx}(\log_a x) = \frac{1}{x}\cdot\frac{1}{\ln a}$

eg $\frac{d}{dx}\left[\log_{10}(ax^2+bx+c)\right] = \frac{1}{ax^2+bx+c}\cdot\frac{1}{\ln 10}\cdot\frac{d}{dx}(ax^2+bx+c)$

$= \frac{2ax+b}{(ax^2+bx+c)\ln 10}$

So

$= \frac{-1}{m}\sum_{i=1}^{m}\left[\frac{y^i}{y'^i}\frac{\partial}{\partial\theta_j}(y'^i) + (1-y^i)\frac{\partial}{\partial\theta_j}\frac{(1-y'^i)}{(1-y'^i)}\right]$

$= \frac{-1}{m}\sum_{i=1}^{m}\left[\frac{y^i\cdot y'^i\cdot(1-y'^i)}{y'^i}\frac{\partial}{\partial\theta_j}\theta^T x + \frac{(1-y^i)(0-(y'^i)(1-y'^i)\frac{\partial}{\partial\theta_j}\theta^T x}{(1-y'^i)}\right]$

$= \frac{-1}{m}\sum_{i=1}^{m}\left[y^i(1-y'^i)x_j^i - (1-y^i)(y'^i)x_j^i\right]$

$= \frac{-1}{m}\sum_{i=1}^{m}\left[y^i - y^i y'^i - y'^i + y^i y'^i\right)x_j^i$

$= \boxed{\frac{1}{m}\sum_{i=1}^{m}(y'^i - y^i)x_j^i}$

$\frac{\partial}{\partial\theta_j}J(\theta)$ Given $J(\theta) = \frac{-1}{m}\sum_{i=1}^{m}\left[y^i\log(y'^i) + (1-y^i)\log(1-y'^i)\right]$

So by chain rule. $\frac{\partial}{\partial y'^i}J(\theta)\cdot\frac{\partial y'^i}{\partial z^i}\cdot\frac{\partial z^i}{\partial\theta_j}$

$\min_{\theta_j} J(\theta)$

$y'^i = \sigma(z^i)$

$z^i = \theta^T x^i = \theta_0 + \theta_1 x_1^i + \theta_2 x_2^i + \cdots$

- Linear regression is line/curve fitting problem.
- Linear regression try to fit curve in the sense as data distributed
- Decision boundary is draw by hypothesis function.

classification using Gradient Descent.

· Logistic Regression (SML)

- Binary classification algorithm
- Implement through supervised machine learning algorithm
- used for binary classification problems
- When we solve classification problems by using some classification algorithm :. actually model. learns its decision boundary for classification by training data in training phase.
- Logistic Regression based on Sigmoid function

Q. Why we not used linear regression for classification?

Ans. One method of classification is to use linear regression (either implemented by simultaneously equation or by Gradient descent) and map/classify all prediction using some threshold lets says greater than 0.5 as a 1 and than less 0.5 as a 0. But this method doesn't work well because classification is not actually a linear function. So we used sigmoid function to introduce non-linearality. Sigmoid function takes any value between $-\infty$ to ∞ and map between 0 and 1. 0, and 1 both are inclusive.

- Hypothesis function should satisfy: $0 \leq h_\theta(x) \leq 1$

So we use sigmoid function also called logistic function to achieve this.

$h_\theta(x) = g(\theta^T x)$ $\quad \theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots$

$z = \theta^T x$. $\quad$ { range $-\infty$ to $+\infty$. linear regression.

$g(z) = 1/1+e^{-z}$

- we start with our old hypothesis (linear regression), except we want to restrict the range to 0 and 1
- This is accomplished by plugging ($\theta^T x$) into logistic function.
- $g(h_\theta(x))$ will give us the probability that our output is 1.
- for example, $g(h_\theta(x)) = 0.7$ gives us the probability of 70% that our output is 1.

$$h_\theta(x) = P(y=1 \mid x;\theta) = 1 - P(y=0 \mid x;\theta).$$
$$P(y=1 \mid x;\theta) + P(y=0 \mid x;\theta) = 1.$$

By default $g(h_\theta(x))$ return the probability to belong to class 1

Logistic Regression Decision Boundary.

- Decision boundary may be of any shape (straight line, curve, circle etc.)
- In order to get our discrete 0 and 1 classification, we can translate the output of hypothesis function as follow:

$$h_\theta(x) \geq 0.5 \rightarrow y=1$$
$$h_\theta(x) < 0.5 \rightarrow y=0$$

- The way our logistic function g behaves is that when its input is greater than or equal to zero, its output is greater than or equal to 0.5.

$$g(z) \geq 0.5 \text{ when } z \geq 0$$
$$g(z) < 0.5 \text{ when } z < 0$$

- So if our input to g is $\theta^T x$ then that means:

$$h_\theta(x) = g(\theta^T x) \geq 0.5 \text{ when } \theta^T x \geq 0$$

- from these statement we can now say:

$$\theta^T x \geq 0 \rightarrow y=1$$
$$\theta^T x < 0 \rightarrow y=0$$

- The decision boundary is the line that separates the area where $y=0$ and where $y=1$. It is created by our hypothesis function.

Let's learn our unknown parameters by training data.

which has 2 input features x_1, x_2.

$$\theta = \begin{bmatrix} 5 \\ -1 \\ 0 \end{bmatrix}$$ so $y=1$ if $\theta_0 + \theta_1 x_1 + \theta_2 x_2 \geq 0$

$$5 + (-1)x_1 + 0x_2 \geq 0$$

$$5 - x_1 \geq 0$$

$$-x_1 \geq -5$$

$$x_1 \leq 5$$ — decision boundary.

- In this case our decision boundary is a straight vertical line place on the graph where $x_1 = 5$ and everything on the left of it ~~that~~ denotes $y=1$ while everything to the right denotes $y=0$.
- Again the input to the sigmoid function $g(z)$ (e.g $\theta^T x$) doesn't need to be linear and could be a function that describes a circle (eg $z = \theta_0 + \theta_1 x_1^2 + \theta_2 x_2^2$) or any shape to fit our data.

Logistic Regression cost function.

- The more our hypothesis is off from y, the larger the cost function output. If our hypothesis is equal to y, then our cost is 0.
- Rate of change is greater when there is greater difference between actual and predicted mean more cost.

- we define cost function for both classes.

So our cost function is

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \text{cost}(h_\theta(x^i), y^i)$$

→ cross entropy.

$\text{cost}(h_\theta(x^i), y^i) = -\log(h_\theta(x^i))$ if $y=1$ — for class 1

$\text{cost}(h_\theta(x^i), y^i) = -\log(1 - h_\theta(x^i))$ if $y=0$ → for class 0.

$\text{cost}(h_\theta(x), y) = 0$ if $h_\theta(x) = y$.

$\text{cost}(h_\theta(x), y) \to \infty$ if $y=1$ and $h_\theta(x) \to 1$

$\text{cost}(h_\theta(x), y) \to \infty$ if $y=0$ and $h_\theta(x) \to 0$

we can compress our cost functions for class 1 and class 0 into one case:

$$\text{cost}(h_\theta(x), y) = -y \log(h_\theta(x)) - (1-y)\log(1 - h_\theta(x))$$

Notice that when $y=1$ then second term $-(1-y)\log(1-h_\theta(x))$ will be zero and will not affect the result. If $y=0$ then the first term $-y \log h_\theta(x)$ will be zero and will not affect the result

So our full version of cost function is

$$J(\theta) = \frac{-1}{m} \sum_{i=1}^{m} \left[y^i \log(h_\theta(x^i)) + (1-y^i)\log(h_\theta(x^i)) \right]$$

Binary cross entropy.

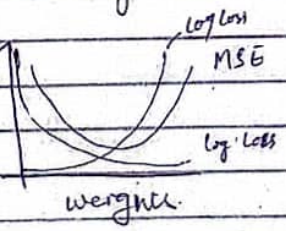
$$J(\theta) = \frac{-1}{m} \sum_{i=1}^{m} \left[y^i \log(y'^i) + (1-y^i)\log(1-y'^i) \right]$$

Binary cross entropy based / built on 0 and 1

→

descent:- Moving down.

→ GD have such behaviour it automatically converged weights.
→ Sigmoid function basically compute the probabilities of classes.
→ cross entropy penalize model more as compared to MSE when prediction deviate from true label during training.
→ Derivative of cost function penalize model direction w.r.t some reference point origin
- Derivative / Gradient of cost function basically tells slope of function (cost function).
- Derivative of cost function w.r.t weight tells either cost function increase or decrease ..... when we increase in weight.



- Derivative / Gradient only tells direction to move.
- GD actually move / push gradient

Logistic Regression (Gradient Descent).

-we use 0.5 as threshold because both classes (0,1) have equal chances to occur mean both classes have same probability (0.5) and sum of probability is 1

steps

① Randomly initialize weights between (0 and 1)

② calculate prediction.

calculate z $z = \theta^T x$

calculate Sig(z)

Assign class to input after thresholding.

calculate accuracy

③ Repeat until desired accuracy achieved using weight update equations $\theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$

Hypothesis function $y'^i = h_\theta(x^i) = \sigma(z^i)$ $\because z = \theta^T x$

$\because \sigma(z) = 1/1+e^{-z}$

Cost function: Binary cross entropy

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^i \log y'^i + (1-y^i) \log (1-y'^i) \right]$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} (y'^i - y^i) x_j^i$$

So weight update equation is

$$\theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} (y'^i - y^i) x_j^i$$

By default vector defines cols wise.

Logistic Regression vectorized implementation.

- For calculating prediction.

$$h_\theta = g(X\theta)$$

and after thresholding we can find y'.

where X is data set with additional input features whose all values equal to 1.

and θ is vector of weights.

- For calculating cost

$$J(\theta) = \frac{1}{m} * -y^T \log(h) - (1-y)^T \log(1-h)$$

- For update weights:

$$\theta := \theta - \alpha \nabla J(\theta).$$

$$\nabla J(\theta) = \frac{1}{m} \cdot X^T \cdot (g(X \cdot \theta) - y).$$

Assignment $\theta := \theta - \frac{\alpha}{m} \cdot X^T \cdot (g(X \cdot \theta) - y)$

not use threshold in weight update equation to update weights but used when you want to calculate accuracy

learning rate for assignment 0.01, 0.1, 0.3, 0.5

Stoping criteria:

→ 1000 iteration print accuracy in each iteration during training

→ accuracy on training as well as on testing data.

→ map 2 and 4 class to 0 and 1.

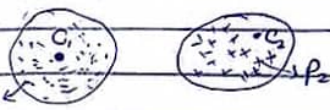
stoping criteria

- is desired accuracy achieved or not.
- No. of iteration
- Thetas / weights converged or not. Not suitable
- minimum loss Not suitable

K-Medoids clustering Algorithm (USML)

- Also called as partitioning around medoid (PAM)
- A medoid can be defined as the point in the cluster, whose dissimilarities with all the other point in the cluster is minimum.
- The dissimilarity of the medoid C_i and object or point P_i is calculated by using $E = |P_i - C_i|$
  Note that other distance measures can be used to calculate dissimilarity.
- The cost in K-Medoids algorithm is given as

$$C = \sum_{C_i} \sum_{P_i \in C_i} |P_i - C_i|$$



C = cost (a numerical number)

Algorithm working

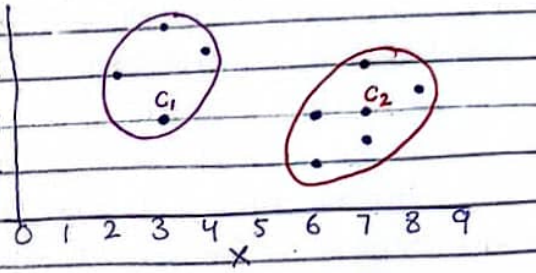
- Initialization of K-random points.
  Select K random points out of the n data points as points the medoid. Medoids must be selected among the data
- Compute the closeness of each data point w.r.t each medoid by using any common distance metric methods
  Associate each data point to the closet medoid
- While the cost decreases.
  For each medoid m, for each data point o which is not a medoid
  ① Swap m and o, associate each data points to the closet medoid, and recompute the cost
  ② If the total cost is more than that in the previous step, undo the swap and stop.
  cost tells us how better clusters are made by these medoids

K-mean is best for numeric data
K-medoid is best for categorical data.

Dry Run.

Lets we have input data that have two input features say X and Y and total data points are equal to 10 and K = 2 (C_1 and C_2). lets $C_1 = (3,4)$; $C_2 = (7,4)$

Sl.No	X	Y	Dissimilarity from C_1	Dissimilarity from C_2
0	7	6	6	2
1	2	6	3	7
2	3	8	4	8
3	8	5	6	2
4	7	4	4	0
5	4	7	4	6
6	6	2	5	3
7	7	3	5	1
8	6	4	3	1
9	3	4	0	4



C_1 cluster		C_2 cluster	
Points	Dissimilarity	Points	Dissimilarity
1	3	0	2
2	4	3	2
5	4	6	3
	11	7	1
		8	1
			9

- So each point is assigned to the cluster of that medoid whose dissimilarity is less
- As we see points 1, 2 and 5 go to cluster C_1 and 0, 3, 6, 7 and 8 go to cluster C_2
- The cost $C = (3+4+4) + (2+2+3+1+1) = 11 + 9 = 20$.
- Now randomly select one non-medoid point and recalculate the cost. let randomly selected point be (7, 3). As this point belong to C_2 so we swap it by C_2.

Sl.No	X	Y	Dissimilarity from C_1 (3,4)	Dissimilarity from C_2 (7,3)
0	7	6	6	3
1	2	6	3	8
2	3	8	4	9
3	8	5	6	3
4	7	4	4	1
5	4	7	4	7
6	6	2	5	2
7	7	3	-	-
8	6	4	3	2
9	3	4	-	-

C_1 cluster		C_2 cluster	
Points	Dissimilarity	Points	Dissimi
1	3	0	3
2	4	3	3
5	4	4	1
	11	6	2
		8	2
			11

Lastly

- Each data point is assigned to that cluster whose dissimilarity is less.
- As we see data points 1, 2 and 5 go to cluster C_1 and 0, 3, 6, 7 and 8 go to cluster C_2
- The cost $C = (3+4+4) + (3+3+1+2+2) = 11+11 = 22$.

Swap cost = present cost - previous cost.

$= 22 - 20$

$= 2 > 0$

As swap cost is not less than 0, we undo the swap. Hence (3,4) and (7,4) are the final medoids. The clustering would be in the following way.

Advantages.

- It is simple to understand and easy to implement.
- K-Medoid Algorithm is fast and converges in a fixed no. of steps
- PAM is less sensitive to outliers than other partitioning algorithms.

Disadvantages.

- The main disadvantage of K-medoid algorithm is that it is not suitable for clustering non-spherical (arbitrary shaped) groups of data points/objects. This is because it relies on minimizing the distance between the non-medoid objects and the medoid - briefly, it uses compactness as clustering criteria instead of connectivity.

Q:- what is meant by algorithm is less sensitive to outliers?

Ans Means algorithm is more robust to outliers and it is less likely to produce extream or misleading results due to the presence of outliers in the data.

K- Medcid Assignment (Information Retsieval System)

Steps

①- Assign id's to each document.

②- Extsact unique words from all docs.
By typecasting in set

③- Make vocabulary of unique words and assign id's
By dictionary

④- Build Term Doc Metsix (TDM)
TDM dimension's = no. of docs x vocabulary size.

⑤- Generate query and make query vector

⑥- Compute matching scose of query and docs

⑦- Retsieve docs on the basis of scores

Dry Run

① docs = ["I live in Lahore", "Lahore city", "Pakistan Lahore live", "Lahore is city of Pakistan"]

② unique_words = {"I", 'live', "in", 'Lahore', 'city', 'Pakistan', 'is', 'of'}.

③ vocabulary = {"I": 0, 'live': 1, 'in': 2, 'Lahore': 3, 'city': 4, 'Pakistan': 5, 'is': 6, 'of': 7}

④ TDM

	0	1	2	3	4	5	6	7
0	1	1	1	1	0	0	0	0
1	0	0	0	1	1	0	0	0
2	0	1	0	1	0	1	0	0
3	0	0	0	1	1	1	1	1

⑤ query = Pakistan city..

0	0	0	0	1	1	0	0

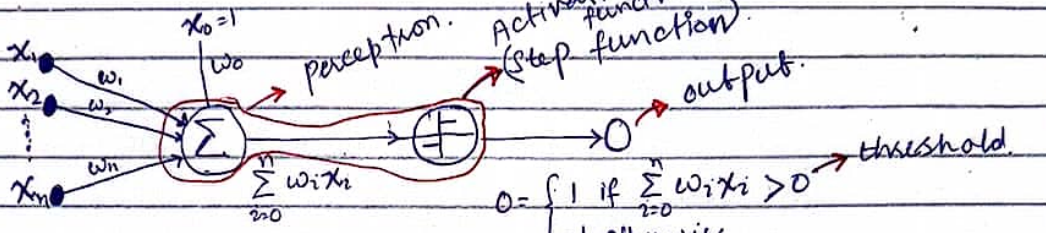
⑥ Matching score = [0, 1, 1, 2]

⑦ Results

doc-4, doc-1, doc-3

Artificial Neural Network

- It is a powerful learning algorithm inspired by how the brain works.
- Perceptron is the basic unit of artificial neural network.
- we use "bias" during learning of artificial neural network so it gave better results on unseen data.



$$O = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 & \text{otherwise} \end{cases}$$

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

$$Z = \underbrace{b + \sum_{i=1}^{n} w_i x_i}_{\text{perceptron.}}$$

$$Z = w \cdot x + b. \quad \text{(vectorize implementation)}$$

Here - $x_1, x_2, \dots x_n$ are input features
- b = bias = $x_0 w_0 = w_0$ $\because x_0 = 1$.
- Assign class 1 or -1 after thresholding
- $w_0, w_1, \dots w_n$ are respective weights of input features
- Step function compare output with threshold.

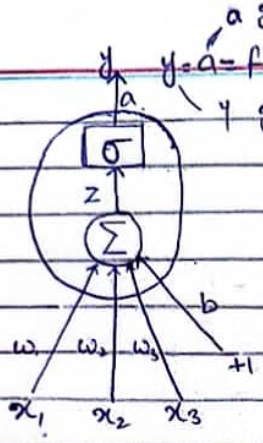
So in this way we build a perceptron for two class problems

Logistic Regression as Perceptron.

- Simply use sigmoid unit/function as activation function to introduced non-linearity.

a generally use for hidden layer.

y generally use for final layer.



$y = a = f(z)$

$z = w_1x_1 + w_2x_2 + w_3x_3 + b.$

$z = \sum_{i=1}^{3} w_i x_i + b.$

$z = w.x + b.$

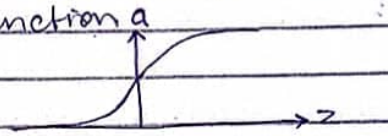
$y = a = f(z)$

As we know loss/cost function for LR is cross entropy.

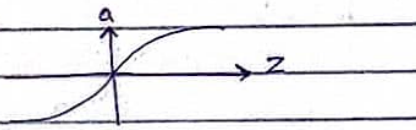
$L_{CE}(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1-y) \log(1-\hat{y})]$

Different Activation function

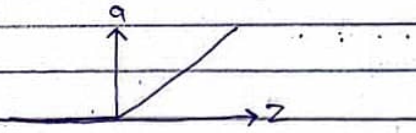
① Sigmoid: $a = 1/1+e^{-z}$



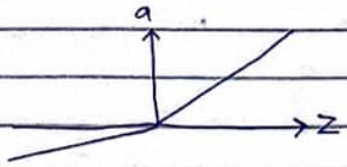
② Tanh: $a = e^{z} - e^{-z} / e^{z} + e^{-z}$



③ Relu: $a = \max(0, z)$



④ Leaky Relu: $a = \max(0.1z, z)$



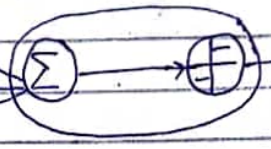
perceptron ≈ Neuron.

- Basic unit.
- No hidden layer

XOR using Astificial Neural Network

AND: First we impliment AND by Astificial Neural Network.

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1



$$y = b + \sum_{i=1}^{2} w_i x_i$$

$$y = b + w \cdot x$$

$$y = b + w_1 x_1 + w_2 x_2$$ (perceptron)

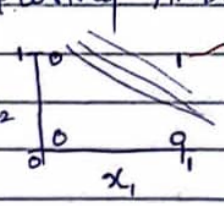
$$y = \begin{cases} 0 & \text{if } b + w \cdot x \leq 0 \\ 1 & \text{if } b + w \cdot x > 0 \end{cases}$$

- So required equations are.

AND

x_1	x_2	y	Required Equations	Simplified	Solution	
0	0	0	$b + w_1 x_1 + w_2 x_2 \leq 0$	$b \leq 0$	b: -0.5	-0.5
0	1	0	$b + w_1 x_1 + w_2 x_2 \leq 0$	$b + w_2 < 0$	w_1: 0.4	0.3
1	0	0	$b + w_1 x_1 + w_2 x_2 \leq 0$	$b + w_1 < 0$	w_2: 0.4	0.3
1	1	1	$b + w_1 x_1 + w_2 x_2 > 0$	$b + w_1 + w_2 > 0$		

- Ploting AND.



As you see there infinitly solutions exist.

- Decision boundary.

we can find decision boundary by $y = b + w_1 x_1 + w_2 x_2$

at $y = 0$ so

$$b + w_1 x_1 + w_2 x_2 = 0$$

$$w_2 x_2 = -b - w_1 x_1$$

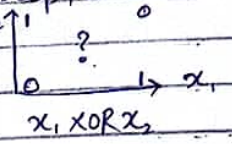
$$x_2 = \frac{-b}{w_2} - \frac{w_1}{w_2} x_1 \rightarrow \text{decision boundary.}$$

This line act as a decision boundary in 2D space.

HTM Theory

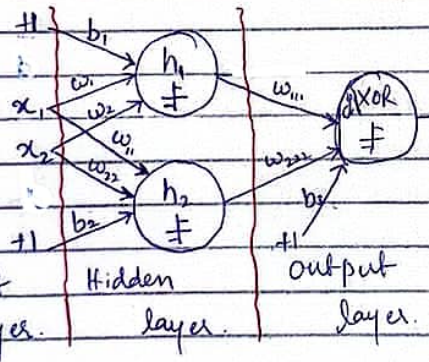
in which the output 0 is assigned to all inputs on one side of the line, and output 1 to all input points lying on the other side of the line.

Similarly we can impliment OR by one perceptron but we can't impliment XOR by one perceptron but with more perceptrones.



Alternative approach for XOR

x_1	x_2	y_1	$h_1 : x_1 \wedge \neg x_2$	$h_2 : \neg x_1 \wedge x_2$	$h_1 \vee h_2$ XOR : y_1
0	0	0	0	0	0
0	1	1	0	1	1
1	0	1	1	0	1
1	1	0	0	0	0



$h_1 = b_1 + \omega_1 x_1 + \omega_2 x_2$

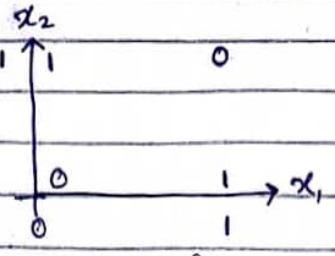
$h_2 = b_2 + \omega_{11} x_1 + \omega_{22} x_2$

$y = \text{XOR} = b_3 + \omega_{111} + \omega_{222} h_2$

- In this ANN there are total 3 perceptrons 2 (h_1, h_2) in hidden layer and 1 (y) in final or output layer.
- They are also called ReLU units.

- There are total 9 unknown weights.
- Input, hidden and output layers may have multiple perceptrons.
- An Artificial Neural Network must have input and output layers but may have 0 or more hidden layers.

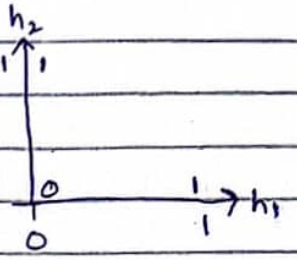
x_1	x_2	$y = XOR$
0	0	0
0	1	1
1	0	1
1	1	0



The original x space.

$h_1 : x_1 \wedge \neg x_2$ $h_2 : \neg x_1 \wedge x_2$ $y_1 = XOR = h_1 \vee h_2$

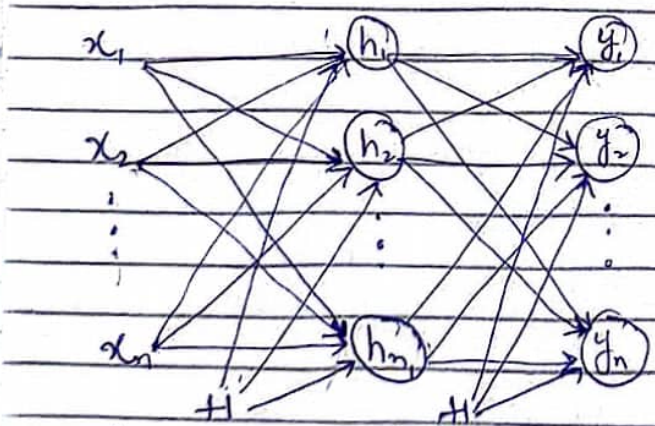
0	0	0
0	1	1
1	0	1
0	0	0



The new h space.

Feed Forward Neural Network

- Previous layer is fully connected with next layer



W and U are weights matrix

W_{ij} of the weight matrix W represents the weight of the connection from ith input x_i to the j^{th} hidden unit h_i.

- Efficient implementation by using weight matrix.
  - Multiply W by x
  - Add bias b.
  - Apply activation function eg sigmoid

$$h = \sigma(Wx + b)$$

Similarly for next layer.

$$z = Uh$$
$$y = \text{softmax}(z)$$

Softmax

→ To compute $P(y=c|x)$ where classes are more than two

→ Softmax takes a logits vector (z) of c (no of classes) asbitrary values and maps them to probability distribution with each value in the range (0 to 1) and all values summing to 1

→ $z = [z_1, z_2, z_3, \cdots z_c]$

→ $\text{Softmax}(z) = \left[\frac{e^{z_1}}{\sum_{k=1}^{c} e^{z_k}}, \frac{e^{z_2}}{\sum_{k=1}^{c} e^{z_k}}, \frac{e^{z_3}}{\sum_{k=1}^{c} e^{z_k}}, \cdots \frac{e^{z_c}}{\sum_{k=1}^{c} e^{z_k}} \right]$

For Example

$z = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]$

$\exp(z) = e^z = [1.822, 3.004, 0.223, 3.320, 24.532, 0.332]$

$\text{Sum of}(e^z) = 33.234$

$\text{softmax}(z) = [0.054, 0.090, 0.006, 0.099, 0.738, 0.010]$

→ Softmax function genesate probability distribution.