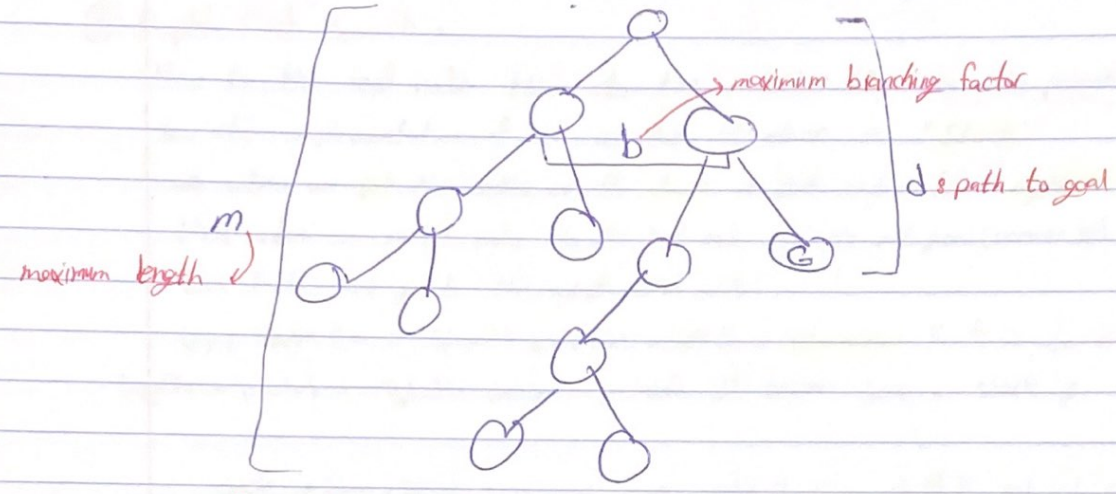


Uniformed search :

① Breadth first Search :-

choose the node from left to righ (level by level)

* على الترتيب الهجائي باليراق

* to implement it on DataStructure we used "Queue"

* space complixity ⇒ b^{d+1}

* time complexity ⇒ b^{d+1}] exponential

⇒ ① complete ② Its not guarntee to be optimal (can not decided)

* the breadth search be optimal when the cost equal in level

② Uniform Cost Search :-

Data structure is "Queue"

- as dijkstra algorithm ⇒ returned the shortest path from initial to goal.
- we calculate the path cost from intial state to the next state.

* الـ node اللي زرناها ما بنزورها . * اذا تكررت node ما بنضيفها مرة

بل بنقارنها مع القديمة (بنختار الاقل cost)

* اذا تساوى الـ cost بنضل على القيد القديمة

* space complexcity ⇒ exponential , time complexity ⇒ exponential

⇒ ① Complete (prevents the cycles) ② Optimal.

* uniform cost special case from Breadth first search (when the cost not equal)

* on the optimality side ⇒ Breadth special case from uniform cost

Because uniform cost is always optimal.

③ Depth First Search :-

Move to the leaf node then do the recursive (Back to the parent)

* the implementation of it as data structure is "Stack"

⇒ when we put the nodes on the stack we put nodes from right to left then when we visited nodes to the leaf node and its not goal (remove it) from stack and go to the next node.

← من ناحية الـ space com. ، الـ depth اغلى من breadth لانه الـ depth يخزن في stack و يعمل remove للـ node و ما بتعمل بالـ memory (ما بحجز مساحة كبيرة)

← بما انه الـ depth يمشي الـ node فممكن الطريق يروح يزورها و يدخل في cycle لذلك ليس معنون أن يكون complete.

* space complexity ⇒ polynomial ($b*m$)
* time complexity ⇒ exponential (b^m)
* Completness ⇒ not gurntee ⇒ not optimal
* Breadth is better than depth in completeness, but depth better than Breadth in space complexity.

× بشكل عام :- ← اذا كان "goal" موجود في اقصى الـ left ← اغلى depth

"the best case in depth & worst case in breadth"

اغلى breadth ← node اذا، level اول "goal" اذا كان ←

"the best case in breadth & worst case in depth"

④ Depth-Limited Search :- similer to depth-first but with limit

the depth limited do recursive but not when reach the leaf node, the recursive condition is not to exceed a specific limit.

* time complexity : b^I , I: depth limit
* space complexity : $b*I$
* Completness :- Not complete
* Optimality : Not optimal

⑤ Herative Deeping :-

* Applies limited depth with increasing depth limit (dynamic)
⇒ Combines advantages of Breadth-first and depth first methods.
* time complexity : b^d (exponential)
* space complexity : $b * d$ (polenomial)
* Completness : Yes it is complete
* optimality : yes it is optimal

⑥ Bi-directional Search : it is not important.
search simultaneously from two directions (forward from initial and backword from the goal)

* Informed Search : uses knowledge to reduce the searching time

- informed search may reach to the optimal solution or may not.

* heuristic :- is function, with input : state, output : estimated cost
↳ from current node to the goal node.
* if I in leaf node ⇒ $h = \infty$
* if I in goal node ⇒ $h = 0$
  * Admissible : if h less or equal the exact cost (Real Cost).
  * nonadmisible : if estimated cost of h large than exact cost.
* if h admissible the solution is guarntee to be optimal.

1 Greedy best first search :

we know that "uniform cost" use path cost, there the Greedy use estimated cost h. (the next state that have less value h)

* Not complete ⇒ can get stuck in loops
* Not optimal
* time complexity ⇒ b^m (exponential)
* space complexity ⇒ b^m
⇒ Data structure = "queue"

[2] A*: use path cost & estimated cost

Data structure is also "priority queue" but sorted by [pathcost + heuristic]

* if h_1, h_2 are admissible heuristic :-

$h_3 = h_1 + h_2 \Rightarrow$ non admissible

$h_3 = h_1 - h_2 \Rightarrow$ non admissible

$h_3 = |h_1 - h_2| \Rightarrow$ admissible

$h_3 = Max(h_1, h_2) \Rightarrow$ admissible

$h_3 = Min(h_1, h_2) \Rightarrow$ admissible

- admissible is better than non admissible, but if there's two admissible which one better? the closer to the exact value

so in the previous exp the best heuirstic is $Max(h_1, h_2)$

* [3] Improving A* (Iterative deeping A*):

A* complete, optimal, fast, but now we should improve space complexity.

to improve the space complexity we will combines between depth & A*.

* we apply the iterative deeping as a cutoff rather than the depth for the iteration.

* cut off value is the smallest cost of any node.
(we work depth but don't reach leaf nodes, the limit is cutoff)

* we going to the left, do recursive (Back) if we reach cutoff (cost) larger than one I define.

* New cutoff value is lower cost in guenrat nodes (not visited)

cutoff value = min (Cost of next visited nodes)

Simple Recursive Best-first Search (RBFS) :-

we will reduce the memory by :-

① if current f-value exceeds (تجاوز) this alternative f-value → backtrack to alternative path.

② Upon backtracking change f-value to best f-value of its children.

* the difference between A^* & RBFS that A^* doesn't remove any node from the memory, but RBFS remove from memory when it move to alternative path.

* RBFS is better about space complexity because it remove from memory
But IDA* better than RBFS, because IDA* works depth

Simplified Memory Bounded A^* :- (SMA*)

Efficiency in memory → SMA* is better than IDA* & RBFS.
but it more complicated and expensive.

* Bounded ⇒ have specific number of states stay in the search space
- At first work as A^* until reach the limit, then start remove worst nodes

* when we reach the limit and remove the worst nodes from the tree we replace them by generated next states.

* SMA* is not guantee to be complete if the number of nodes in the path greater than memory bound.

↳ so to reach the solution, the memory bound should be at least equal to the path

Local Search & Problem Optimization:

Ex:- "Traveling Salesperson Problem" (TSP)

ساعي البريد وظيفته توصيل الرسائل الى مكانها المحدد ، والترتيب الذي يتبعه لا يهم المهم فقط ان تصل جميع الرسائل الى أماكنها. (يتم بالتسلسل وليس دفعة واحدة)

← يقدر احل هاي المشكلة بخوارزمية A* لكن يحتاج مساحة ضخمة لكل ما يستخدمها ، بنلجأ لـ optimization.

* Optimization: Improved existing solution but there is no optimality.
- any local search is complete but space complexity is constant beeause there is no path.

* some problems, the path isn't important (we don't need it). In this case I can remove it from memory ⟹ space comp. will be order of constant.
↳ this type of search is informed Search (still have/use knowledge)
* Time complexity, we can not know it (depends on algorithm & intial state)
* Completness ⟹ It has no meaning here.

* Improving Search Method:-
- make algorithms more efficient (improve time & space).
  - avoiding repeated states
  - utilizing memory efficiently.
- use addional knowledge about the problem.
  - properties (shape) of the search space (more interstige area are more investigated first)
  - pruning of irrelevard areas.

* In "TSP" problem ⟹ we will use the goodness: total of all the distances that we crossed them until we reach to the our destination.
* goodness is knowledge but not heuristic ⟹ there is no optimality.

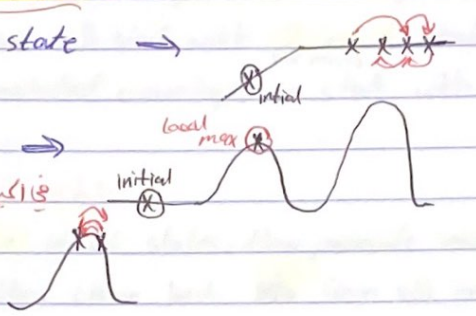
* Key Idea (surprisingly simple)
1. Select (Random) initial state.
2. Make local modifiration to improve current state (evaluate nore to other states)
3. Repeat step 2 until goal state found. (or out of time)

Problems in local search :

① enter steady state ⟹ (loop)

② local Maximum ⟹ في اكبر منها ولكن حواليها من الـ max

③ ridges. ⟹



[1] Hill climbing Search :-

* Hill climbing variations :-
  - Random walk hill climbing [chose the state randomly]
  - Random restart hill climbing [repeat the algorithm more than 1 time] ↳ change ikial state
  - Hill climbing with both.

Random walk hill climbing :-

choose next state randomly, in this way we can go to the worst case but we can avoid the local minimum.

[2] Simulated Annealing :-

it is hill climbing search but it choose the next state as greedy approach.

* the gerral idea : if next state better than current state then go to it. else, go to it under a Condition.
* Condition ⟹ $R < e^{\left(\frac{-\Delta E}{k_B T}\right)}$

* start initially with high temperature, explore next steps randomly.
* over time, reduce the temperature to explore less.
* focus on exploitation and best choices when temperature is cold.

at $T=0$ ⟹ return current_state (الأفضل)

else : next_state = random_step (current_state, T)

(next-current) ← current

لـ بعد ما نطلع معنا قيمة من الـ random ← بنطرح قيمة اللي طلعت معنا منها الـ current

اذا كانت اكبر من صفر فبنعتمدها وبنخليها الـ current

اذا لا بنعتمدها برضو ولكن مع الاحتمالية لقبولها واللي هي بتعتمد على الحرارة

(بالاول بتكون الحرارة عالية وتكون نسبة الـ random عالية وبتقل مع الوقت)

* Population Based Algorithms (local search Algorithms)

In these algorithms we will start with 2 initial states (or more), but in hill climbing and simulated annealing we start with one intial state.

[1] Local Beam Search :-

At first generate 'k' initial states, then generate next stat from all intial states, then choose best 'k' from all next states.

* using k steps ⇒ make expand on search space (fully search space)
⇒ this way reduce the local minimum problem.

[2] Genetic Algorithm :-

in genetic algorithm, At first should determine the chromosome of problem (① length of it in terms number of genes, ② the implemntation of genes, ③ the order)

if we read the chromosome from left to right or right to left, does it make sense or not?

* the chromosome is the most important thing in formalization.

— طول الكروموسوم هو عدد الجينات ويتكون حسب السؤال ممكن مثلاً اعراض

— الكروموسوم يمثل حل للذي اذا وجد كروموسوم اخر يعني مثلاً حلين

* we choose the first 2 chromosomes (solutions) randomly.

* to generate a chromosome ~~Bit~~ with type of implemtation is binary, use random function for each index in the chromosome.

if rand() >05 put the value 1

if rand() <0.5 put the value 0

① The first step is generating process by create set of chromosomes randomly.

② Then choose the best two chromosomes to matching between them.
↳ choose these two parents depends on function called "Fitness Function"

③ The matching between two parents called 'crossover'
the crossover between two parents gives two child.

* **Fitness function** :-

is evaluation function returns a value indeato to the best chromosome.

↳ fitness function like heuristic

بس ما اعطاه الـ heuristic لانه هو بيختار افضل chrom. من ناحية الـ quality مش cost

* There's no reference in genetic because the genetic is optimization techniques.

* Any optimization techniques its **termination** is :

① time out ② reached to the requirment needed ③ There is no improvement

* in hill climbing & simulated annealing we use the random walk,

in genetic we will use the 'mutation'. (to exit from the steady state)

* **Mutation** means :- way to exit the non optimaization mode - if the implementation of chromosome is binary choose any gene and **reflect** its value.

* Contraint Satisfaction Problem (CSP)

- **Formalization** :-

given a set of variables, each variable takes at least one value.

since this assigment must satisfy all constraints.

* exp. :- **map coloring**.
  - constraints : adjacent regions must have different colors.
  - [Red, Green, Blue] → Domain (Values)
  - The regions (Countries) → Variables

* **In general** : The CSP solution is incremental approach (choose variable & make assignment) then go to the next state (variable) until reach the goal state.

* **Depth** is the **best algorithm** for CSP, because the depth do 'Backtracking' but in these problems it doesn't reach the leaf.

⇒ it make backtracking when there is a problem in the constraints.

* **make backtracking in two cases** :-

① When constraints are violated.

② Reach to the variable doesn't have any values that I can choose.

⇒ So, the algorithm that will use it is "Depth With Backtracking"

* the genrated search space is very very large.
  space complexity ⇒ $[N! \times D^{N}]$

* there is no cycle problem in this algorithm, because it use incremental approach. so it is impossible to assign twice.

* There is no optimality in CSP.
* We should improve time complexity by reduce the backtracking and reduce the backtracking by use the heuristic, by choose the best order of variable assigned & the selection of values is also important.

* heuristic by: [1] The order of variables:

(mrv) — Most-Constrained (remaining) variable (minimum remaining value)
variable with the fewest possible values is selected to minimize the branching factor. اختيار المتغير الذي يحتوي اقل عدد من القيم الممكنة له

(mcv) — Most_Constring variable
variable with the largest number of constraied on other unassigned variables.
⇒ in graph choose node with largest number of links.
اختيار المتغير الذي يتفاعل بشكل اكبر مع باقي المتغيرات

[2] The selection of values for the variables:-
(LCV) — Least Constrainening value.
for a selected variable, choose the value that leaves more freedom for future choices. تختار القيمة التي توفر بشكل اقل على المتغيرات الاخرى

* Forward checking: to reduce backtracking, we remove all values from the neighbor nodes that caused the conflicts with the assignment after assign the current node.
forward ⇒ because I look to the nodes that unassigned.
forward checking ⇒ looks (to) the neighbors nodes (there is constrant) only.

* Adversaria Search & Games :

the searching techniques in games are separated from other searching algorithms, this due to games qualities.

⇒ ① The search space of games is very large.

② In games there are two players (agents).

③ In the environment, there is a chance & there is nothing fixed.

* Pruning : when a state is excluded then we already excludes all next states from this state.

* Formalization : ① intial state depends on game

② Sucessor function : legal moves depends on game

③ utility function : produces a value for terminal state

* There is No optimal solution in games.

* Minimax Algorithm : " Depth first Search"

The first one choose the Max and the second choose the Min of Firsts Max.

* with Alpha-Beta : ⟹ to delete some parts of search space (reduce time & space)

utility function

Alpha : Max player Beta : Min player

① The pruning happens between Max & Min. (Max makes a pruning for Min vice versa) but there is no pruning between Max & Max or Min & Min.

② The decision of pruning comes from high level.

③ The node can make the pruning if it has a value less or equel alpha.

* - Beta cutoff pruning : alpha (childs)
(search in max)

- Alpha cutoff pruning : alpha (parent)
(search in min)

Machine learning:

Is set of techniques that are classified to 3 types:-

1. supervised learning ⇒ correct answers for each example
2. unsupervised learning ⇒ correct answers not given
3. Reinforcement learning ⇒ occasional rewards

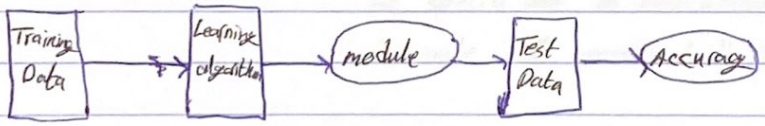
* machine learning is **independent on input**.
because we convert the input to the formula that all machines learning understand it.
this formula is **feature vector**.

* **Feature vectors** - is an array of one dimension, it representation all inputs in data.

* **Data set** is the input of machine learning.

Decision Tree:

* divide the data into two groups. First part for training & second part for testing.
* the decision tree that I will get from learning is set of rules [set of if-statements]



* we need automatical method to build generalized tree, so you should choose the most discriminative features.

* the most important attribute is the root. (that have more information)

* $E(S) = -P_1 \log_2(P_1) - P_2 \log_2(P_2)$ ← if 2 — the entropy

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

information gain ⇒ $G(S,Q) = E(S) - \sum_{i=1}^{k} P_i E(S,Q_i)$

⇐ بنوخد كل attribute من الجدول وبنحوش احتمالياته وبنحسبلهم الـ entropy (E)
بعدها بنطلع الـ information gain للداتا كلها وبقارن مع باقي الـ attributes وبنوخد الروت اعلى G
وبنضل نكرر نفس الطريقة على كل node لحد ما يصير كلهم داخلها نفس الشي

✱ Evaluation :-

The available data set is divided into two disjoint subsets:-
- the training set (for learning a model)
- the test set (for testing the model) ⇒ holdout set

مشتركين او بس بنقسمهم لمجموعتين

✱ holdout set ⇒ نقسم الداتا لجزئين مثل فوق ، هاي بنستخدمها للداتا الكبيرة

✱ n-fold cross-validation ⇒ on common n = 10 or 5
⇒ data is partitioned into n equal size disjoint subsets
⇒ when data not large
⇒ يعمل test على الواحد والباقي يعمل عليهم training وهكذا
⇒ فائدته انه مرر على كل الداتا بالرغم من انه بطيء

✱ Leave-one-out cross validation ⇒ when the data is very small
⇒ special case of cross-validation
⇒ single test example & all the rest used in training

② ✱ Second type of evaluation :- Validation set : divided into 3 subsets
- a training set (60%)
- a validation set (20%) ⇒ estimating parameters in learning algo.
- a test set (20%)

✱ Classification measures: ① Accuracy = عدد الـ samples اللي تنتماج / مجموع الـ samples في الـ set

✱ اذا كانت الداتا imbalance يعني واحد من class عنده الغلبة ما بتعرفه.

*② Confusion matrix:

	classified Positive	Classified Negative
Actual Positive	TP (true positive)	FN (False negative)
Actual Negative	FP (false positive)	TN (true negative)

⇒ **Precision p** :-

$$p = \frac{TP}{TP+FP}$$

فيعني انت صنفت صح من البوزتف ,

(معنية بس بالpositive) ↙ دقة تصنيف العناصر الايجابية

⇒ **Recall r** :-

$$r = \frac{TP}{TP+FN}$$

↙ اكتشاف جميع العناصر الايجابية الموجودة

كأنه كم رجع صح من الداتا كلها.

* حسب كل حالة ممكن يتغير اي واحدة فيهم الاحسن ولكن اذا احترنا نستخدم :

→ **F value (F-score)** :-

$$F = \frac{2pr}{p+r}$$

* Really (actual) positive = TP + FN
  Really (actual) negative = TN + FP

* **Accuracy** :

$$\frac{TP+TN}{TP+FP+TN+FN}$$

* **Overfitting** :-
  Low Bias ⇒ يعني نسبة الخطأ قليلة في ال training
  Higher Variance ⇒ نسبة الخطأ كبيرة في ال testing

الحل هو التحكم في ال Degree of Polynomial

① **Prepruning** : اذا ال feature ال gain تاعه وصل رقم معين خلص وقف ما تكمل بناء ⇒ easier, faster

② **Post pruning** : ابني كل ال tree وبعدها باشر قص لحد ما توصل للمطلوب. → works better

Naive Bayes Algorithm:

1 Learning Phase:

جدول تكرار بالـ frequency – جدول لكل attribute

ex:-

Outlook	Play=Yes	Play=No
Sunny	2/9	3/5
overcast	4/9	0/5
Rain	3/9	2/5

9 yes (all) $\Rightarrow p(yes) = \frac{9}{14}$

5 No (all) $\Rightarrow p(No) = \frac{5}{14}$

Wind	Yes	No
strong	3/9	3/5
weak	6/9	2/5

Temperature	Yes	No
Hot	2/9	2/5
Mild	4/9	2/5
Cool	3/9	1/5

2 Test Phase:

- Given new instance, predict its result.

X = (outlook=Sunny, Temperature=Cool, Humidity=High, Wind=strong)

نطلع احتمالية كل attribute بالنسبة لـ Yes / No

P(outlook=sunny / Play=Yes) = 2/9

P(outlook=sunny / play=No) = 3/5

P(Temerature=cool / play=Yes) = 3/9

P(Temperature=cool / play=No) = 1/5

$\Rightarrow$ $P(Yes/X) \backsim [P(sunny/yes)\,P(Cool/yes)\,P(High/yes)\,P(strong/yes)] \cdot P(play=yes)$

$P(No/X) \backsim [P(sunny/No)\,P(Cool/No)\,P(High/No)\,P(strong/No)] \cdot P(No)$

$\Rightarrow$ the biggest number is the answer.

adv:
- Fast to train (fast to classify)
- Not sensitive to irrelevant features
- Handles real & discrete data
- Handles streaming data well.

disadv: assumes independence of features

* Artificial Neural Networks (ANN) :-

* if the problem linear seprable (بقدر أفصل ال 2 class بخط مستقيم) we need just 1 layer.
* if it Nonlinear seprable usually we need 2 or more layers.
  ↳ ما بقدر أفصل بينهم

* # input neurons = # features (لأنو كل neurons بكون مشبوك مع واحد)
  #output neurons = # classes (if class >2)

* # middle neurons not fixed (depending on the problem) كل ما كانت المشكلة معقدة اكثر بنحتاج عدد اكبر من neurons

* Activation function :

$$*\left[\sum x_i \cdot w_i - \theta\right] \leftarrow \text{step}$$

we use this in one layer system. (1 , S≥0) (0,S<0)

* قبل ما ابلش training لازم اكون عارفة عدد ال neurons وعدد ال layers ونوع ال activate function واختيارنا لل weight و threshold ال بشكل random.

* The Perception :- (one layer)

$$\Rightarrow \left[e(p) = Y_d(p) - Y(p)\right] \quad \text{where } p = 1,2,3,\ldots$$

(predict: $Y_d(p)$; actual: $Y(p)$ — الي طلع معنا بالحسابات بتاعتي)

$$w_{New} = W_{old} + \Delta \Rightarrow \left[W_N = W_o + e \times \alpha\right]$$

(e = error)

- if the error is positive → increase perception output Y(p)
- if the error is negative → decrease Y(p)

① Initalisation :- set initial weights ($w_1, w_2, \ldots w_n$) & threshold θ Random (-0.5, 0.5)

② Activation : activat Y(p) by appling inputs ($X_1(p), X_2(p) \ldots$) & desired output $Y_d(p)$.
  بنطبق ال activat function.

③ Weight training :- update the weights as the above

④ Iteration :- Increase iteration p by one, go back to step 2 & repeat.

Multilayer neural networks:

The back-propagation training algorithm.

1 Initialisation: set all weights & threshold to Random $\left(\frac{-2.4}{F_i}, \frac{2.4}{F_i}\right)$

F_i: total num of inputs of neuron j.

2 Activation activation func.:- sigmoid $[\Sigma x w - \theta]$

$$\delta_k(p) = \frac{d Y_k(p)}{d X_k(p)} * e_k(p)$$

error gradient $$\delta_k(p) = Y_k(p) * [1 - Y_k(p)] * e_k(p)$$

where $$Y_k(p) = \frac{1}{1 + \exp[-X_k(p)]}$$

3 Weight training

after calculate the error gradient for the neurons,

calculate the weight corrections:- $\Delta w_{ij}(p) = \alpha x_i(p) \cdot \delta(p)$

$\Rightarrow$ update the weights:- $w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$

* sigmoid $\Rightarrow$ random value in range (0, 1)

Tricks: ① replace the sig function with tangent function

$$Y^{tanh} = \frac{2a}{1 + e^{-bx}} - a \qquad a, b: \text{constants}$$ ⇒ faster

generalised delta rule ← ② accelerate training by including a momentum term in delta Rule ⇒ less noise

$$\Delta w_{jk}(p) = \beta \cdot \Delta w_{ik}(p-1) + \alpha \cdot y_i(p) \cdot \delta_k(p)$$

$(0 < \beta < 1)$

learning with adaptive rate ③ α بين كل مرحلة واذا ظلت ثابتة بزيد α واذا اختلفت بنقص α بنقص e $\Rightarrow$ $0 < e < 1$

Nearest Neighbor:

1-Nearest Neighbor (very simple)

label a new point the same as the closest known point.

* Two-dimensional ⇒ $Dist(a,b) = sqrt\left[(a_1-b_1)^2+(a_2-b_2)^2\right]$

* Multivariate ⇒ $Dist(a,b) = sqrt\left[\Sigma(a_i-b_i)^2\right]$

K-Nearest Neighbor

if the problem Not numarical we use Similarity metric (Num. of matching attri.)

لما يعطيني Problem جوية بتتوزع على اقرب الحالات الها.

فبنحط حسب كل sample عدد الاختلافات بينه وبين المطلوب وبنوخذ اقل اشي منهم بعدد k

مثلاً لو كان k = 3 وكانوا التنتين نفس النتيجة خلص نفسها مبس اذا كانوا مختلفين

بتوخذ نتيجة الاقرب.

* Increase K ⇒ Makes KNN less sensitive to noise

Decrease K ⇒ Allows capturing finer structure of space, sensitive to noise.

Curse of Dimensionality:-

- Irrelevant attributes easily "swamp" information from relevent attributes.

Unsupervised Learning: (Clustering)

* Cluster:- A collection of data objects/points such that:
  - similar (related) to one another in same group.
  - dissimilar (unrelated) to the objects in other groups.

* Cluster Analysis:-

Find similarities between data according to characteristics underlying the data & grouping similar data objects into clusters.

* Clustering quality :

- Inter-clusters distance (maximized) ⟹ different cluster بين ال
- Intra-clusters distance (minimized) ⟹ points at same cluster ال بين ال

* Data Types :-

- Discrete Feature

Has only a finite set of value eg zip codes, rank or set of words in a collection of documents.

- Continuos Feature

Real numbers as feature values eg temperature, height or weight

* Data Representations :-

- Data matrix (object-by-feature structure)

$$\begin{bmatrix} x_{11} & \cdots & x_{1f} & \cdots & x_{1p} \\ \vdots & & \vdots & & \vdots \\ x_{i1} & \cdots & x_{if} & \cdots & x_{ip} \\ \vdots & & \vdots & & \vdots \\ x_{n1} & \cdots & x_{nf} & \cdots & x_{np} \end{bmatrix}$$

* data (n) points with p dimensions.
* Two modes: row & column for different entities
* Document/word matrix

- Distance/dissimilarity matrix :- (object-by-object structure)

$$\begin{bmatrix} 0 & & & \\ d(2,1) & 0 & & \\ d(3,1) & d(3,2) & 0 & \\ \vdots & \vdots & & 0 \\ d(n,1) & d(n,2) & \cdots & \cdots \end{bmatrix}$$

* n data points but registers the distance
* A symmetric/triangular matrix
* single mode: row & column for same entity

* Distance Measures :

* Cosine Measure ⟹

$$Cos(X,Y) = \frac{X_1Y_1 + \cdots + X_nY_n}{\sqrt{X_1^2 + \cdots + X_n^2} + \sqrt{Y_1^2 + \cdots + Y_n^2}}$$

$$d(X,Y) = 1 - Cos(X,Y)$$

* K-means Clustering:

* Partitionize Clustering Approach:-

$$E = \sum_{k=1}^{k} \sum_{x \in C_k} d^2(x, m_k) \quad , \quad d^2(x, m_k) = \sum_{n=1}^{N} (x_n - m_k)^2$$

* نقسم الداتا عندي لـ k clusters و بنوخذ مجموع المسافات بين كل نقطة والـcenter تبعها ، وبنوخذ الاقل.

⇒ heuristic method.

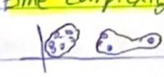
1. Initialisation ⇒ بنختار نقاط تكون center بشكل عشوائي
2. Assign each object to the cluster of the nearest seed point (distance) ← بحسب المسافات
3. Compute the new seed points ⇒ بفحص النقاط الحالية center اذا هم فعلاً صح أولا
4. بعد اختار center يحقق المسافة بين اي جواته
5. Stop when no more new assigment.

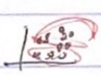
* Stopping/convergence criterion:-

1. no (or minimum) re-assignments of data points to different clusters. (الكلسترز ما بتغيروا)
2. no (or minimum) change of centroids. ⇒ (السنترز ما بتغيروا)
3. minimum decrease in the sum of squared error (SSE) ⇒ الايرور يكون اقل من رقم معين

$$SSE = \sum_{j=1}^{k} \sum_{x \in C_j} dist(x, m_j)^2$$

adv. * simple, very fast, time complexity = $i \cdot k \cdot n$ (linear)

disadv. * - outlier (حالة شاذة مش كثير بتصير) 

- اختيار الـ center بشكل Random ممكن يأثر على تقسيمة المجموعات
- اذا ما كان شكل العينات بشكل دائري ما بكون سلس مع باقي الاشكال 

Logic & Automated Reasoning

→ **Knowledge-based agents**: set of sentences in a formal language.

Propositional logic: "Syntax"

- **Atomic sentence**: A proposition symbol representing a true/false statement
- **Negation**: If P is a sentence, ¬P is (negative) sentence
- **Conjunction**: If P & Q are sentences, P ∧ Q is (and) sentence
- **Disjunction**: If P & Q are sentences, P ∨ Q is (or) sentence
- **Implication**: If P & Q are sentences, P ⇒ Q is (if P then Q) sentence
- **Biconditional**: If P & Q are sentences, P ⇔ Q (if P then Q & if Q then P)

$P \Rightarrow Q = \neg P \vee Q$ (True P ... True ... False Q ...)

$P \Leftrightarrow Q$: P = Q ... True ...

"Semantics":

- **Interpretation**: specifies the true/false status of each proposition. (2^n) n: # symbols
- **model**: the interpretation that output True.
- Rules for evaluating truth with respect to a model:
  - ¬P is true iff P is false
  - P ∧ Q is true iff P is true and Q is true
  - P ∨ Q is true iff P is true or Q is true
  - P ⇒ Q is true iff P is false or Q is true
  - P ⇔ Q is true iff P ⇒ Q is true and Q ⇒ P is true

P ⊕ Q: has 4 interpretations & 2 models (...True ...)

- $(\alpha \Rightarrow \beta) = (\neg\beta \Rightarrow \neg\alpha)$
- $(\alpha \Rightarrow \beta) = (\neg\alpha \vee \beta)$
- $(\alpha \Leftrightarrow \beta) = ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$
- $\neg(\alpha \wedge \beta) = (\neg\alpha \vee \neg\beta)$
- $\neg(\alpha \vee \beta) = (\neg\alpha \wedge \neg\beta)$ → de Morgan
- $(\alpha \wedge (\beta \vee \gamma)) = (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$
- $(\alpha \vee (\beta \wedge \gamma)) = (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$