XGB...

↳ Dynammic Programming

↳ State - vars

↳ Decision - vars.

↳ objective fn → A dynamic fn that associates current rewards with other rewards.

⇒ The complication is we need to evaluate all the decision-variables for all combination of STATE - VARIABLES

⇒ Decision variables + state-variables are interdependent for majority of times.

For t^{th} day till T Days

↳ For i^{th} piece of cake.

↳ Evaluate for all choices.

Max value. ← Take statistic of all those evaluations.

$v(s, t)$ = max Σ Profit across all choices + Profit (At $t+1$)

Because our stop criterion is at $(T+1)^{th}$ day.

XGBoost

KNAPSACK in a looping manner.

⇒ 0/1 → Choice. (लें की हैं)

⇒ States → ° number of elements left in the choice bag
+ ° Weight of Knapsack left.

⇒ Objective → Maximize value (Reward)

⇒ Stop-criterion → If Wt of knapsack ≤ 0 or If elements in choice bag = 0.

⇒ Procedure Flow :-
For all combination of states

\# of items. × Integral wts

Maximize Value of ~~choices~~ Reward over choices.

⇒ Choices + Reward fn follows a dynamic behaviour

Reward at a particular state - Wt = Wi and element checked = #j
↳ depends on the previous states.

K = [[0 for x in range(wt + 1)]
for x in range(n+1)]

K = np.zeros(shape = (n+1, wt+1))

↳ used for caching.

for i in range(n + 1)
for j in range(w+1)

Start $i = 0$
$j = 0 \rightarrow 1, 2, \ldots$

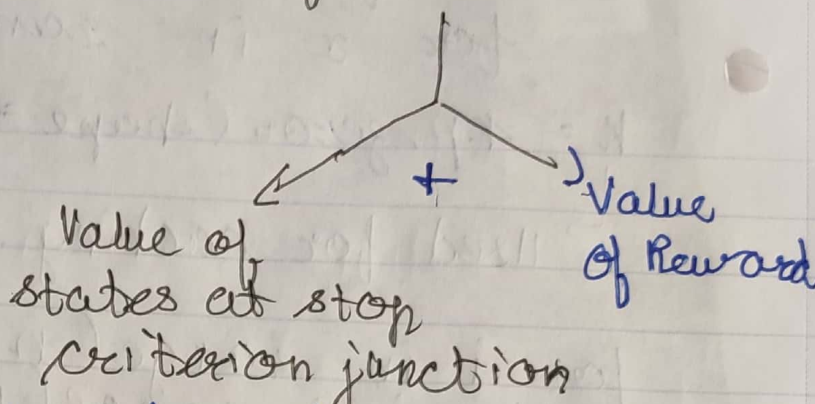
$i=0, j=0$ → stop criterion pivot
↳ $i=0$ or $j=0$.

proc. - Choice based procedure

proc (0,0) → store in. K (0,0)
proc(0, 1)
⋮
⋮ proc(1, 1) → will use previous values of proc

Beleiving that ↲ distribution if the state never changes ~~why~~ the value at i^{th} j^{th} the will always remain the same.

∴ Determination of STOP criterion



IS important.

→ Then we can build a top-down or Bottom-up approach for all combination of state variables.

→ Cache them and use it for future analysis.