1. Adaboost (Adaptive Boosting)

▷ In-depth Math intuition

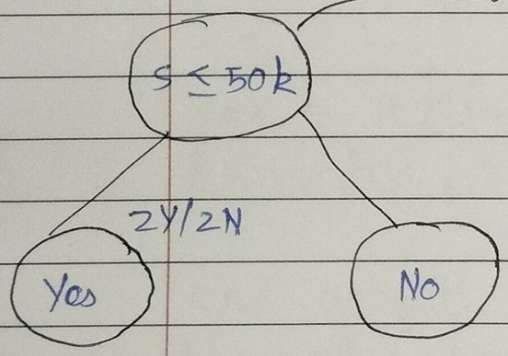
⇒ Dataset

Salary	Credit	Approval
<= 50k	B	No
<= 50k	G	Yes
<= 50k	G	Yes
> 50k	B	No
> 50k	G	Yes
> 50k	N	Yes
<= 50k	N	No

⇒ **Step 1**: We have to create a decision ~~stump~~ stump and we ~~select~~ the best stump

? What is Decision tree stump

↳ A decision-tree stump is a **one-level decision tree** that makes a prediction using a single split on one feature, typically via a simple rule like "if feature ≤ threshold then class A else class B



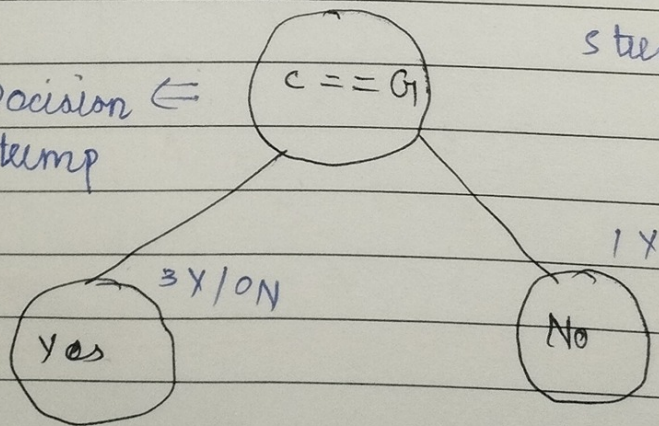
} we call them **weak learners**

we connect weak learners sequentially

:- Another decision-tree

Best Decision ⇐ tree stump

based on purity split



Q = { Which decision-tree stump we would select? }

} This depth-1 tree is (M_1)

Page No.
Date

* To select the best decision-tree stump we use

1. Entropy

↳ formula :- $H(S) = -\sum_{i=1}^{m} p_i \log_2(p_i)$

↳ for k classes explicitly

⇒ ~~$H(S) = -(p_i \log_2 p_i$~~

⇒ $H(S) = -(p_1 \log_2 p_1 + p_2 \log_2 p_2 + \dots + p_k \log_2 p_k$

2. Gini Index (Impurity)

. $Gini = 1 - \sum_{i=1}^{m} (p_i)^2$

(expanded version :- $Gi = 1 - (p_1^2 + p_2^2 + \dots + p_k^2)$

3. Information Gain (This tells us how much uncertainty is reduced if we split the data using a particular feature)

↳ formula :- $Gain(S, i) = H(S) - \sum \frac{|S_v|}{|S|} \times H(S_v)$

∴ $H(S)$:- Entropy of root node

$|S_v|$:- size of each subset v

$|S|$:- total size of the dataset

$H(S_v)$:- Entropy of each subset

⇒ **Step 2:**

- We assign a sample weight

↳ Now how do we assign a same sample weight to all instances in a feature ?

- SW ↳ sol :- Divide all by total for 7 rows (Sample Weight) 1/7 ≅ → for all seven instances (rows)

Dataset

⇒

Salary	Credit	Approval	SW
<=50k	B	No	1/7
<=50k	G	Yes	1/7
<=50k	G	Yes	1/7
>50k	B	No	1/7
>50k	G	Yes	1/7
>50k	N	Yes	1/7
<=50k	N	No	1/7

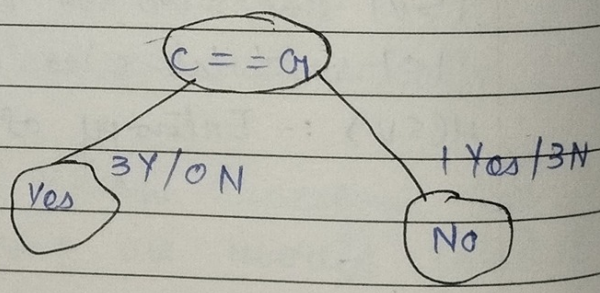
⇒ **Step 3**: Sum of Total Errors & Performance of stump

Dataset

Salary	Credit	Approval	SW
<=50	B	No	1/7
<=50	G	Yes	1/7
<=50	G	Yes	1/7
>50	B	No	1/7
>50	G	Yes	1/7
>50	N	Yes	1/7
<=50	N	No	1/7

Best Decision - tree stump

M1 tree ←



⇒ Credit is N ⇒ so it goes on the leaf in that its yes or no!

→ We know its Yes but depth-1 tree always uses the maximum for prediction. Since we have more No (3) than Yes (1) in No leaf. It will wrong predict it as No

What a stump does ?

↳ A decision stump is a depth-1 tree : it chooses one feature and a threshold (or a category test) to split the training set into two leaves, then assigns each leaf the maximum (majority) label of the samples that fall into that leaf. This is why it's called as "Weak learner"

⊛ Very Important ★★★

↳ Decision Tree Stump Prediction Rule

Example Breakdown :-

Scenario :- A new data point falls into a leaf node with

1 "Yes"

3 "No"

Prediction :- The stump predicts "No" because

- No (3) > Yes (1)
- Majority Wins

★ How it Works :-

1) Step 1 :- Stump splits data based on one feature

- Feature : Age > 30 ?

1 - Yes branch → [1 Yes, 3 No] → Predicts "No"

1 - No branch → [5 Yes, 2 No] → Predicts "Yes"

Key Point

↳ Even if there's 1 "Yes" in the group, the stump ignores it and goes with the maximum count (3 "No") ⇒ This is called as Plurality Vote

When it predicts wrong (depth-1 trees) we calculate sum of total errors

↳ To calculate this we need to find out how many records are predicted wrong

If predicted wrong is 1

then sum of TE = $\frac{1}{7}$

↳ The formula :- used to calculate sum of Total errors

Total Error = Σ (weights of misclassified samples)

↳ In simple words :- Add up the weights of all wrong predictions

If two instances are predicted wrong and its weight is 0.30 and 0.25

↳ Total Error = 0.30 + 0.25 = 0.55

⊛ Performance of stump

↳ To calculate (Performance of stump), we use this formula, which AdaBoost uses internally

→ Performance of stump = $\frac{1}{2} \ln \left[\frac{1 - TE}{TE} \right]$

→ Calculation of Performance of stump using Total Error

$$\text{Performance of stump} = \frac{1}{2} \ln\left[\frac{1 - TE}{TE}\right]$$

$$= \frac{1}{2} \ln\left[\frac{1 - 1/7}{1/7}\right]$$

$$= \frac{1}{2} \ln[6] \approx 0.896$$

∴ ln = Natural logorithm

Eg ln(1) = 0

ln(2) = 0.693

ln(3) = 1.099

ln(10) = 2.303

Different input → Different output

? Where we use Performance of stump?

Performance of stump = 0.896

· Boosting Algorithm (AdaBoost) :- Function

↳ $F = \alpha_1(M_1) + \alpha_2(M_2) + \ldots + \alpha_n(M_n)$

$\alpha_1 = 0.896$ ⇒ Weights

Step 2 :- Update the Weights

⇒ Step 4 :- Update the weights for correctly and incorrectly classified points

Dataset

Salary	Credit	Approval	SW
<=50k	B	No	1/7
<=50k	G	Yes	1/7
<=50k	G	Yes	1/7
>50k	B	No	1/7
>50k	G	Yes	1/7
>50k	N	Yes	1/7
<=50k	N	No	1/7

We know that this one is wrong predicted :-
(So we update weights)

↳ Why update weights ?

The Big Idea :

↳ Force the next stump to focus on samples the previous stump got Wrong

Simple Example :-

You have 5 students taking a test :-

Test 1 Results :

- Students A, B, C → Passed
- Students D, E → Failed

↳ What do you do for Test 2 ?

↳ Don't give equal attention to everyone

✓ Give more attention to D and E (they need help!)

✓ Give less attention to A, B, C (they're doing fine)

Same in AdaBoost:

- Stump 1 Results:
  - samples 1, 2, 3 → Correct
  - samples 4, 5 → Wrong

For Stump 2:

- Increase weight :- for Wrong Prediction
  (sample 4, 5) → "Focus here"
- Decrease weight :- for Correct Prediction
  (sample 1, 2, 3) → "These are easy"

★ Why?

↳ so each new stump fixes the mistakes of the previous stump!

Weight updation formula :-

↳ for correctly classified points

= weight × $e^{-\text{performance of stump}}$

or

New weight = old weight × $e^{-\alpha}$

↳ for wrong predictions

New weight = old weight × $e^{\text{performance of stump}}$

or

New weight = old weight × e^{α}

Where:

$e = 2.718$ (Euler's number)

alpha (α) = performance of stump

Eg for correctly classified points

- new weight = old weight $\times e^{-\alpha}$

$$= \frac{1}{7} \times e^{-(0.896)}$$

$$= 0.058$$

Eg for misclassified points

- new weight = old weight $\times e^{\alpha}$

$$= \frac{1}{7} \times e^{(0.896)}$$

$$= 0.349$$

Dataset (this is how it looks after weight updation)

↳ Updated Weights (Dataset)

Salary	credit	Approval	SW	Updated Weights
<=50k	B	No	1/7 ↓	0.058
<=50k	A	Yes	1/7 ↓	0.058
<=50k	A	Yes	1/7 ↓	0.058
>50k	B	No	1/7 ↓	0.058
>50k	A	Yes	1/7 ↓	0.058
>50k	N	Yes	1/7 ↑	0.349
<=50k	N	No	1/7 ↓	0.058

this is our misclassified point

↳ Now how we gonna tell the next model - that this is wrong prediction point and focus more over here.

↳ Answer :- B in Assignment

⇒ step 5 :- Normalize weights computation and assigning bins

	salary	credit	Approval	SW	Updated weights	Normalized weights
	<= 50k	B	No	1/7 ↓	0.058	0.083
	<= 50k	G	Yes	1/7 ↓	0.058	0.083
	<= 50k	G	Yes	1/7 ↓	0.058	0.083
	> 50k	B	No	1/7 ↓	0.058	0.083
	> 50k	G	Yes	1/7 ↓	0.058	0.083
	> 50k	N	Yes	1/7 ↑	0.349	0.500
	<= 50k	N	No	1/7 ↓	0.058	0.083
Total				1	0.697	1

* Normalize weights means :- making sure all weights add up to 1 (or 100 %)
↳ After ~~after~~ updating weights, they might sum to something weird like 2.5 or 0.8
∴ We want : sum of all weights = 1

* The Formula
Normalized Weight = Weight / sum of all weights

* Bin Assignment ?
→ Bin Assignment is how we select samples for training the next stump based on their weights
→ It forces the next stump to train more on high-weight (misclassified) samples.

Bin Assignment = Groups

Salary	credit	Approval	SW	Update weights	Normalize	Bin Assig
<=50k	B	No	1/7	0.058	0.083	0.000 - 0.083
<=50k	G	Yes	1/7	0.058	0.083	0.083 - 0.166
<=50k	G	Yes	1/7	0.058	0.083	0.166 - 0.249
>50k	B	No	1/7	0.058	0.083	0.249 - 0.372
>50k	G	Yes	1/7	0.058	0.083	0.372 - 0.455
>50k	N	Yes	1/7	0.349	0.500	0.455 - 0.955
<=50k	N	No	1/7	0.058	0.083	0.955 - 1.038
Total			1	0.697	1	1

misclassified point.

↳ Now How the Algorithm automatically picks misclassified points.

⇒ Lets break this down super simple

1. The computer generates random numbers between 0 & 1
→ random_number = random.uniform(0, 1)
like 0.234, 0.789, etc.

2. How does it know which sample to pick automatically?
→ step-by-step Process (super simple):
setup - one time only:
↳ The algorithm creates cumulative ranges (bin) based on weights

# Your normalized weights
weights = [0.083, 0.083, 0.083, 0.083, 0.083, 0.500, 0.083]

Algorithm automatically calculates cumulative sum

cumulative = [0, 0.083, 0.166, 0.249, 0.372, 0.455, 0.955, 1.0]

Size

8.3%

8.3%

8.3%

8.3%

8.3%

50%

8.3%

Now it has ranges :-

Sample 1 :- 0.000 - 0.083

Sample 2 :- 0.083 - 0.166

Sample 3 :- 0.166 - 0.249

Sample 4 :- 0.249 - 0.372

Sample 5 :- 0.372 - 0.455

Sample 6 :- 0.455 - 0.955 ← HUGE!

Sample 7 :- 0.955 - 1.000

The Loop (Automatic Selection) :

lets say we need 100 samples for training :-

```
selected_samples = []

for i in range(100): # Loop runs 100 times

# step 1 :- Generate random number (AUTOMATICALLY!)
rand = random.uniform(0, 1)

# step 2 :- Check which bin this falls into (AUTOMATIC!)
if 0.000 <= rand < 0.083:
    selected_samples.append(1) # Pick sample 1

elif 0.083 <= rand < 0.166:
    selected_samples.append(2) # Pick sample 2

elif 0.166 <= rand < 0.249:
    selected_samples.append(3) # Pick sample 3
```

elif 0.249 <= rand < 0.372:
    selected_samples.append(4) # Pick sample 4

elif 0.372 <= rand < 0.455:
    selected_samples.append(5) # Pick sample 5

elif 0.455 <= rand < 0.955: # ← HUGE RANGE!
    selected_samples.append(6) # Pick sample 6

elif 0.955 <= rand <= 1.000:
    selected_samples.append(7) # Pick sample 7

⇒ Step 2 :- Select data points to the Next Model (Stump)

Real Example (100 iterations):

— Let me show you what happens:

Iteractions	Random num	Falls in Range	Sample picked
1	0.234	0.166 - 0.249	3
2	0.678	0.455 - 0.955	6
3	0.012	0.000 - 0.083	1
4	0.521	0.455 - 0.955	6
5	0.789	0.455 - 0.955	6
6	0.145	0.083 - 0.166	2
7	0.601	0.455 - 0.955	6
⋮	⋮	⋮	⋮
100	0.822	0.455 - 0.955	6

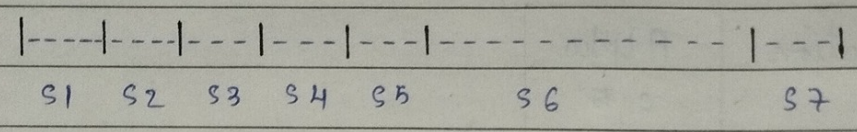
After 100 picks:

- Sample 6 picked - 50 times
- Sample 1 picked - 8 times
- Sample 2 picked - 8 times

- Why Sample 6 gets Picked more?

→ Because its bin is BIGGER!

Imagine throwing a dart at this board:



↳ If you throw randomly, where will you hit MOST?

→ Sample 6's area! It's HUGE

* Simple Analogy:

Lottery Machine:

- Put 1000 balls in a machine
- 83 balls labeled "Sample 1"
- 83 balls labeled "Sample 2"
- 500 balls labeled "Sample 6"
- 83 balls labeled "Sample 7"

- Spin and pick randomly → You'll get "Sample 6" Most often!
- Same logic! The computer does this automatically

⇒ Step 7 :- Final Prediction in AdaBoost

Simple Example :-
4 stumps vote on a new sample :-

Stump	Prediction	Alpha
1	Yes	0.5
2	Yes	0.3
3	No	0.4
4	No	0.2

① (Direct Comparison) :
Yes score = 0.5 + 0.3 = 0.8 ⇒ Winner
No score = 0.4 + 0.2 = 0.6
Predict : Yes !

② (Sign Function) :
Formula :
→ Final Score = $(\alpha_1 \times pred_1) + (\alpha_2 \times pred_2) \ldots\ldots (\alpha_n \times pred_n)$

Final = ~~sign~~ 0.5(Yes) + 0.3(Yes) + 0.4(No) + 0.2(No)
= 0.5(+1) + 0.3(+1) + 0.4(-1) + 0.2(-1)
= 0.5 + 0.3 − 0.4 − 0.2
= 0.2 (positive)

Sign Function
If Final Score > 0 → Predict "Yes"
If Final Score < 0 → Predict "No"
⇒ final score (0.2) > 0 → Yes