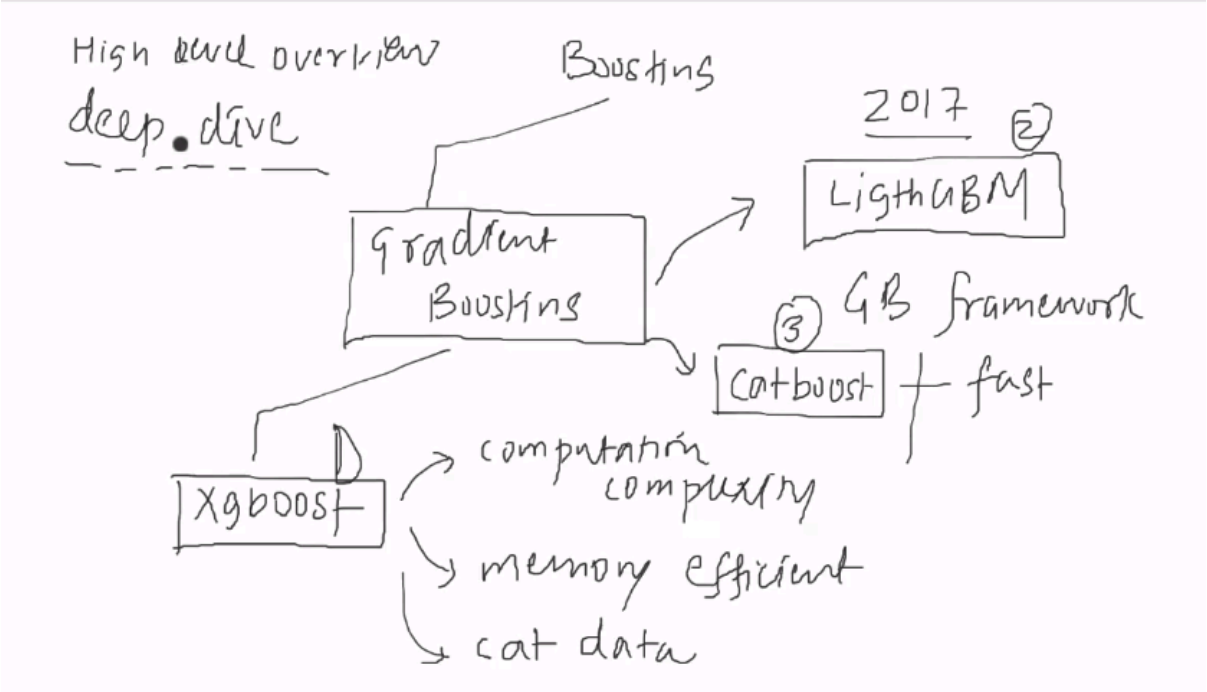
Plan of Attack

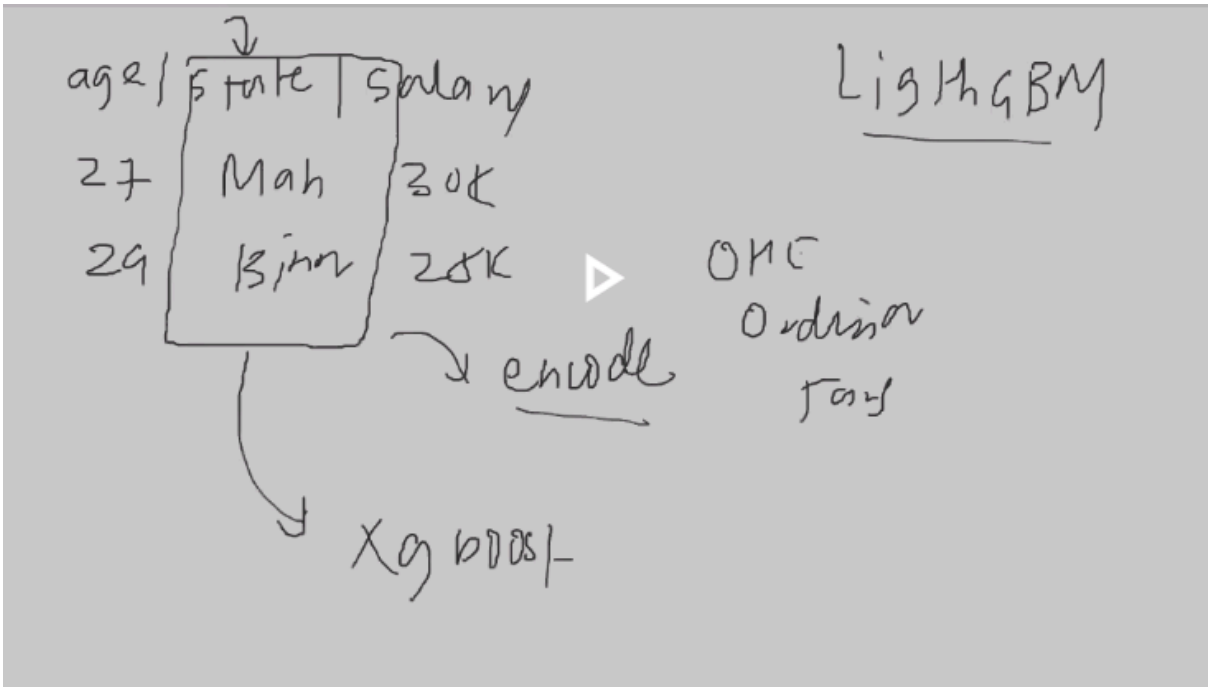


Introduction

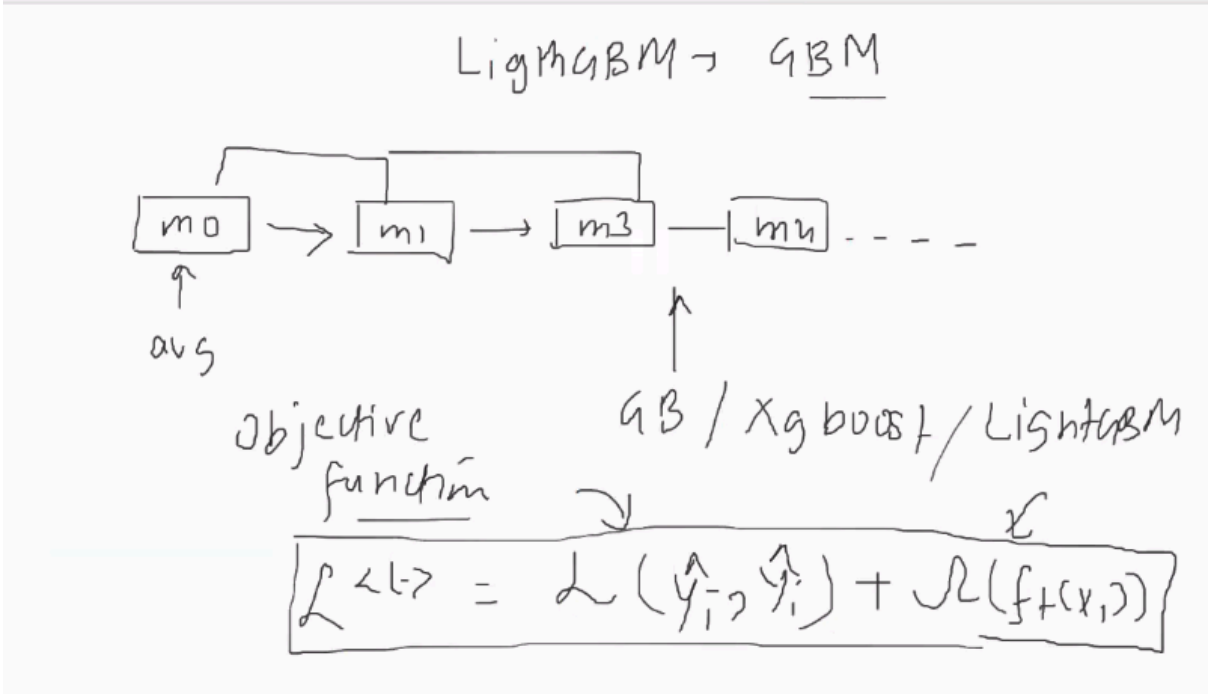
13 April 2024 08:29

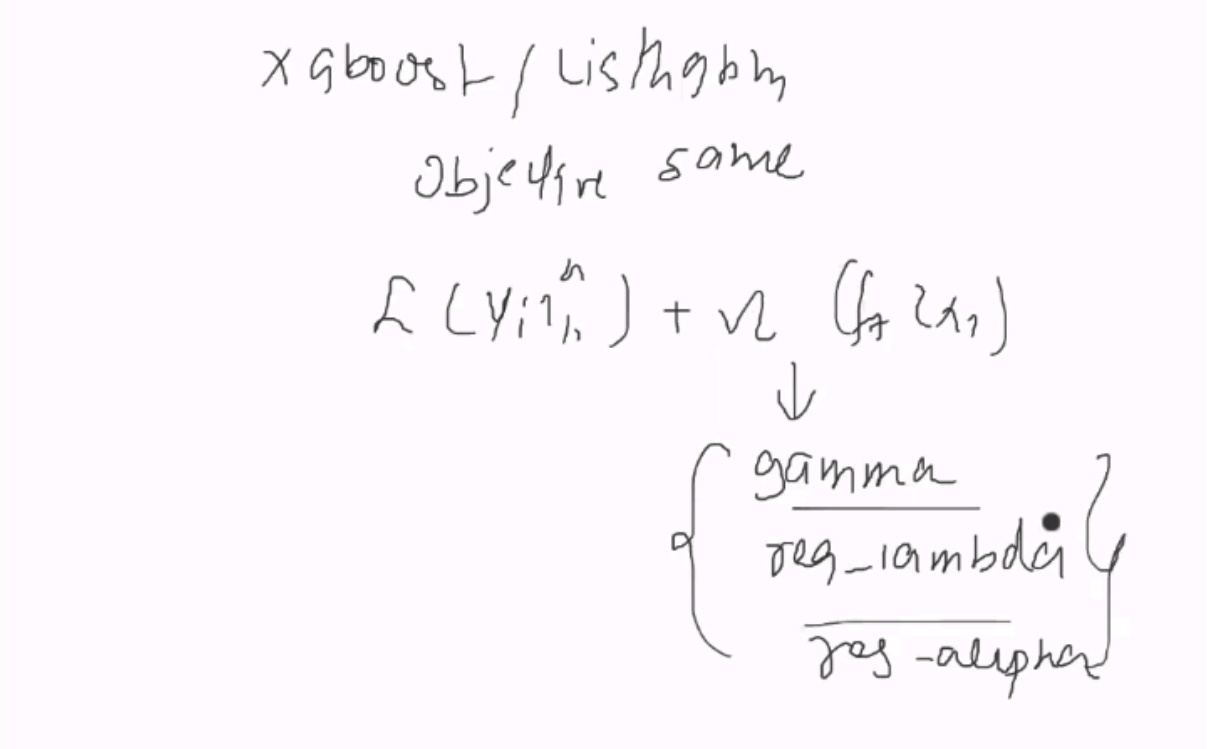
NS

- LightGBM stands for Light Gradient Boosting Machine
- It is an open source Gradient Boosting Framework
- Developed by Microsoft DMTK in 2016
- Goal was to develop high performance Gradient Boosting framework optimized for speed.
- Core Features
  - Speed and memory efficient
  - Accuracy
  - Works with categorical data
  - Multiple Language support
  - Supports Distributed Computing
  - Supports GPU
  - Can handle missing values and sparse data
  - Can work with custom loss functions

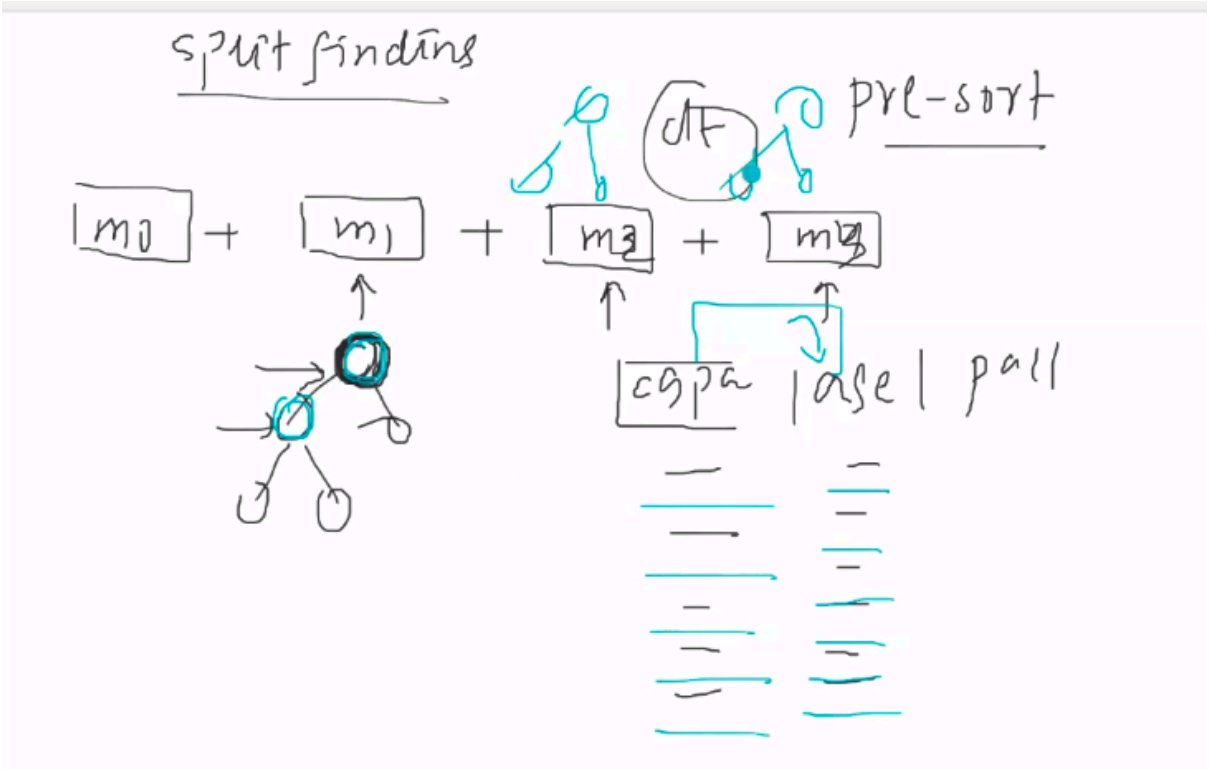


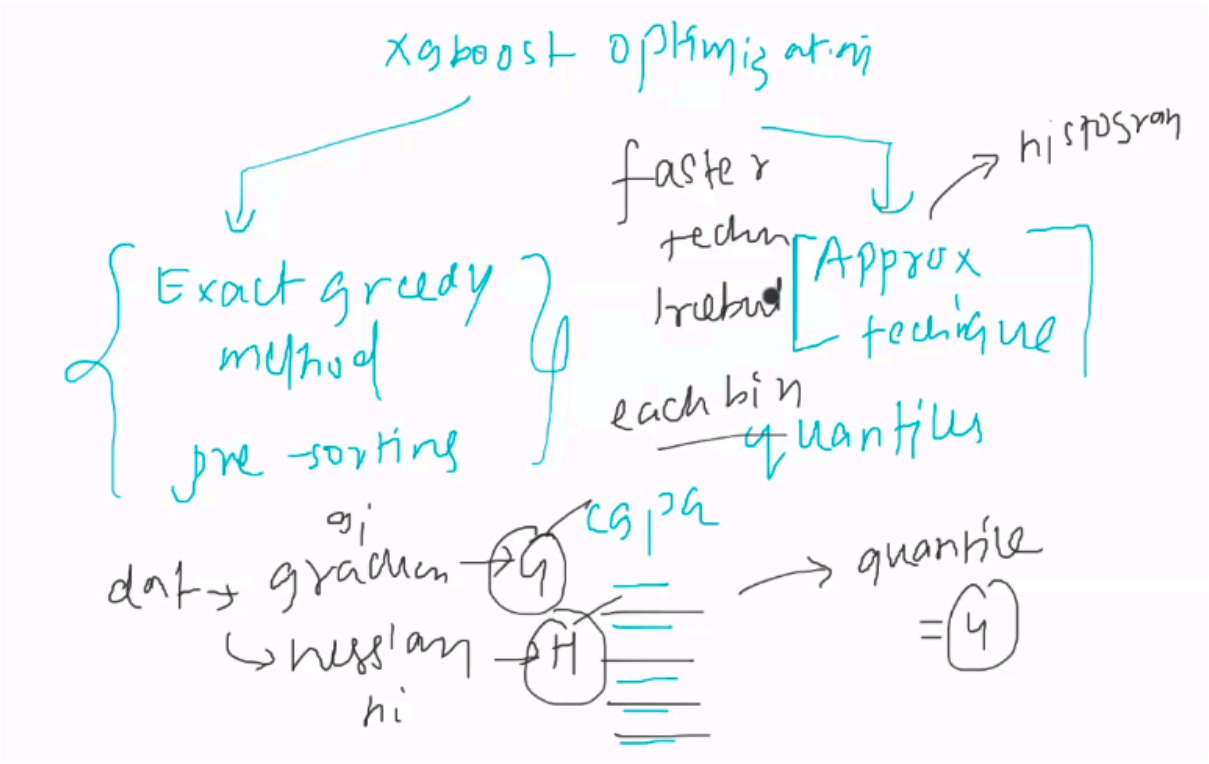
1. Boosting and Objective Function

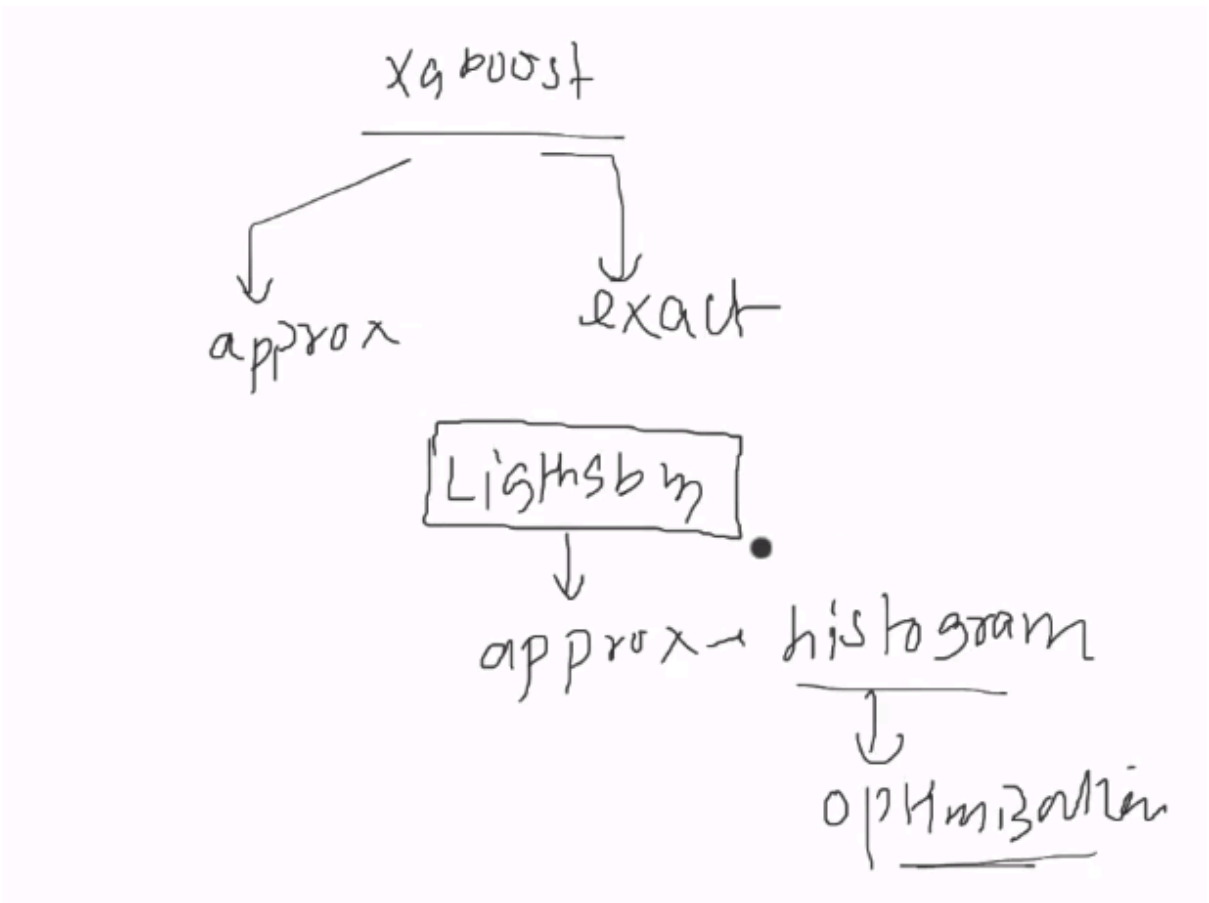




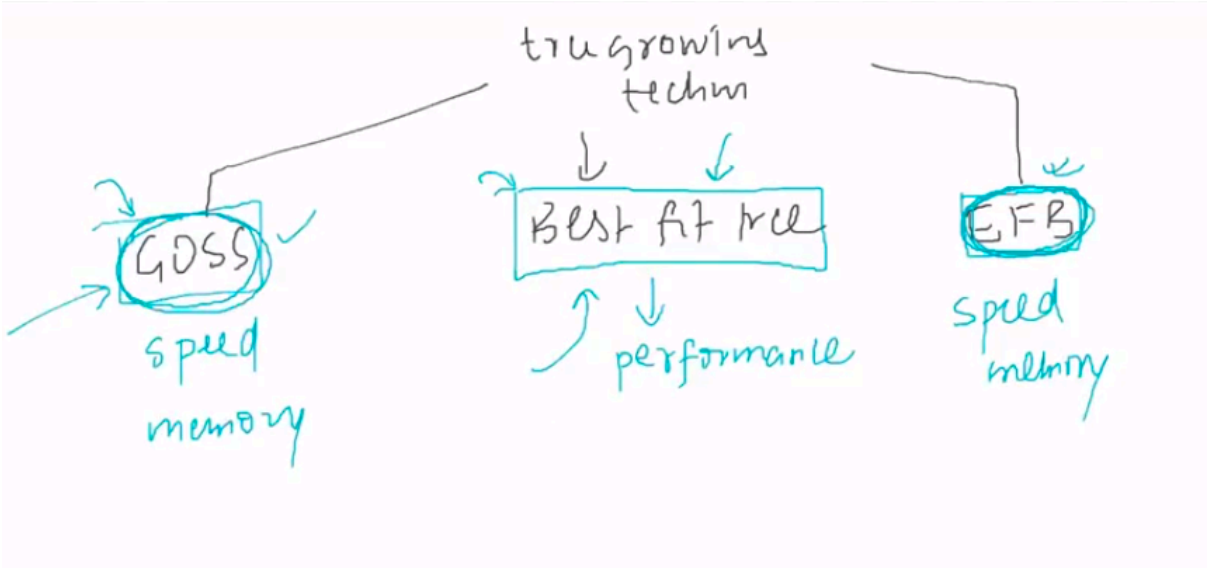
2. Histogram-based Split Finding

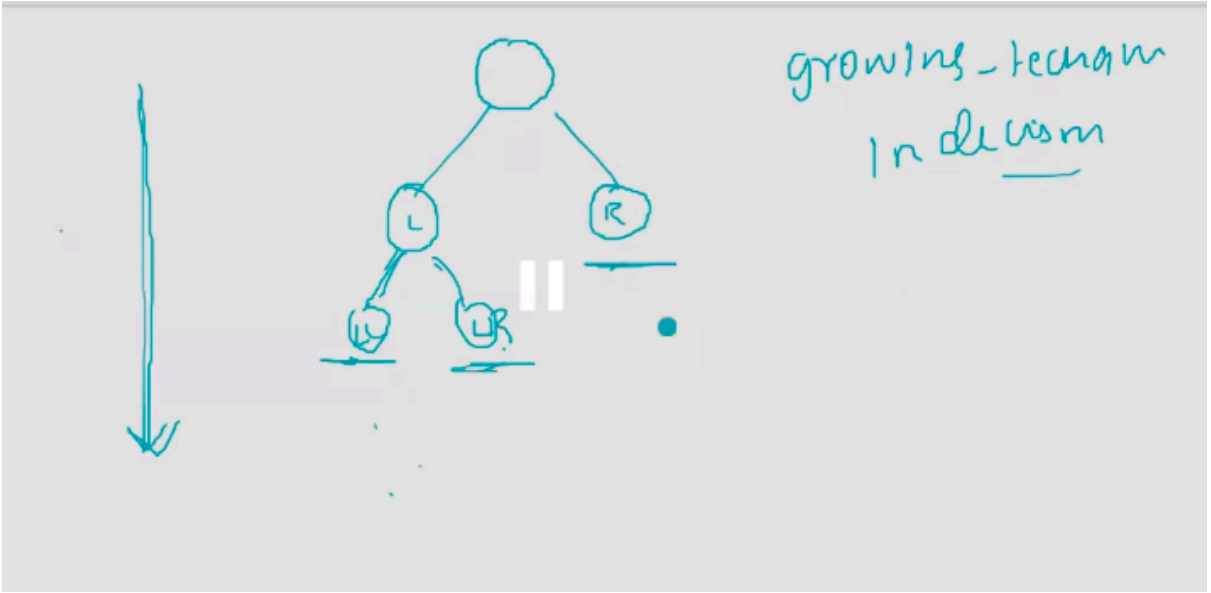


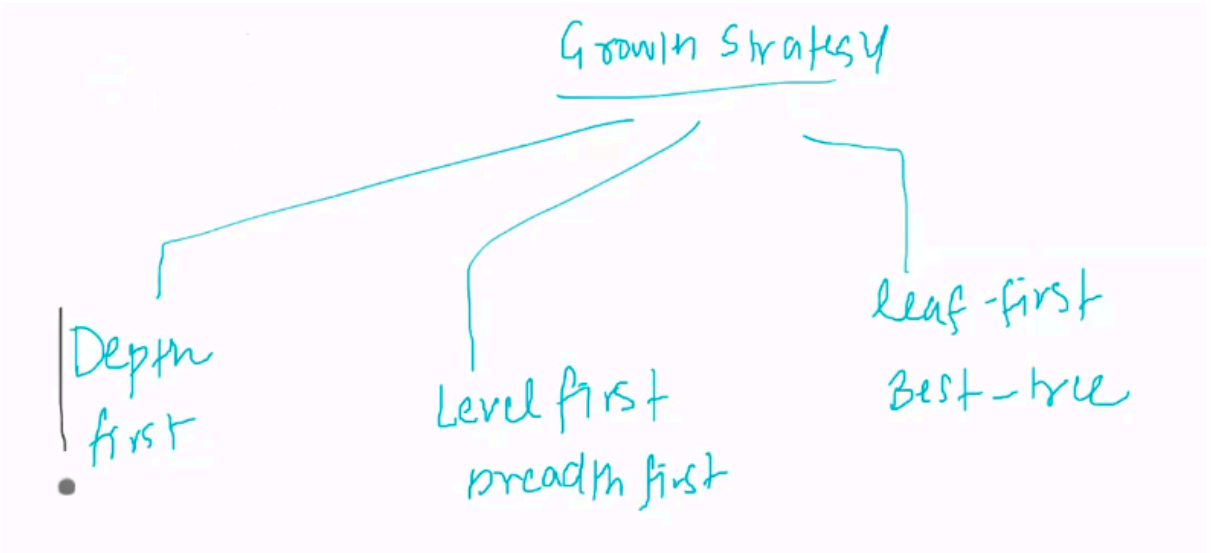


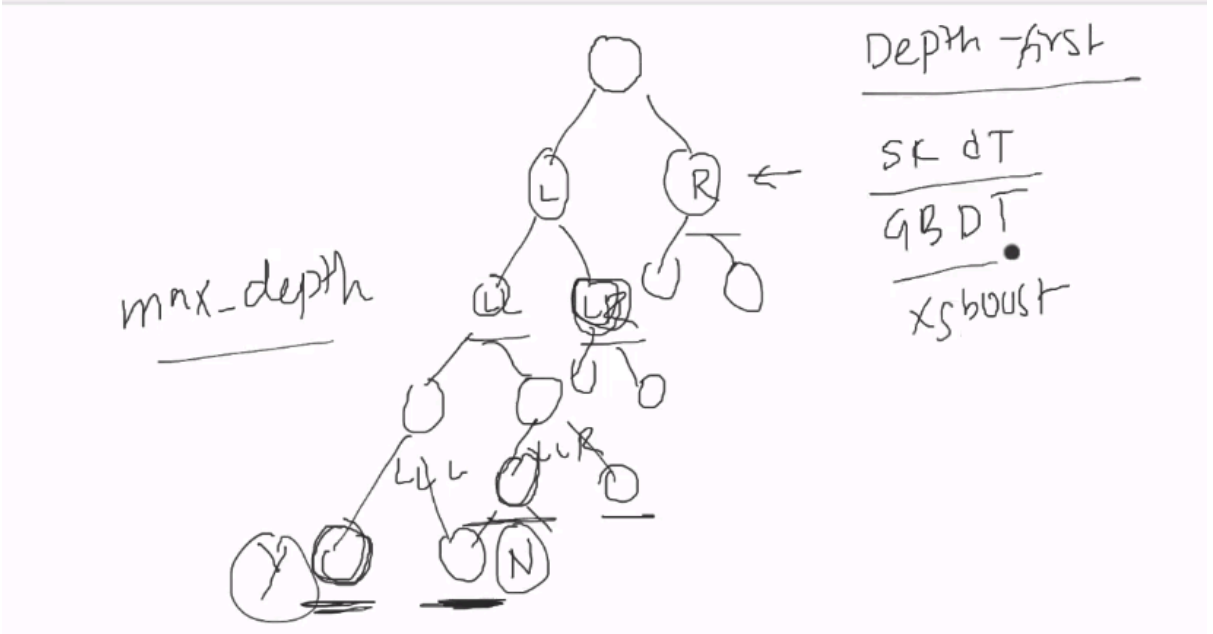


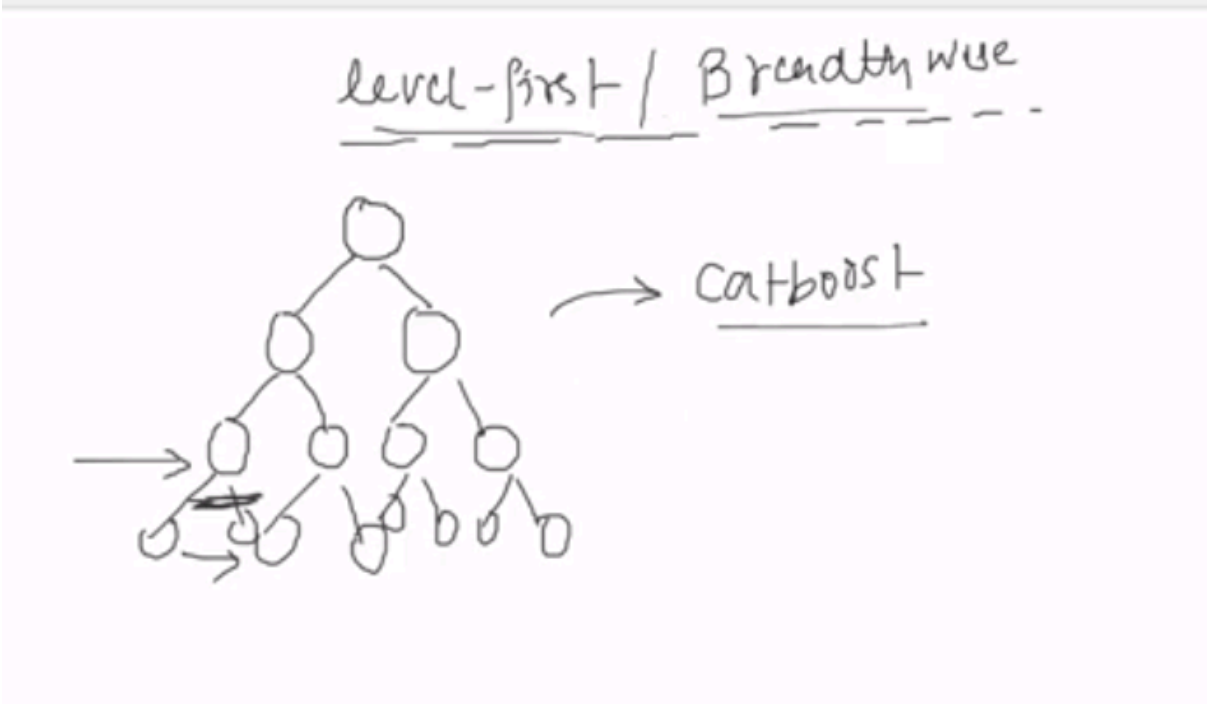
3. Best-fit Tree (Leaf-wise growth strategy)

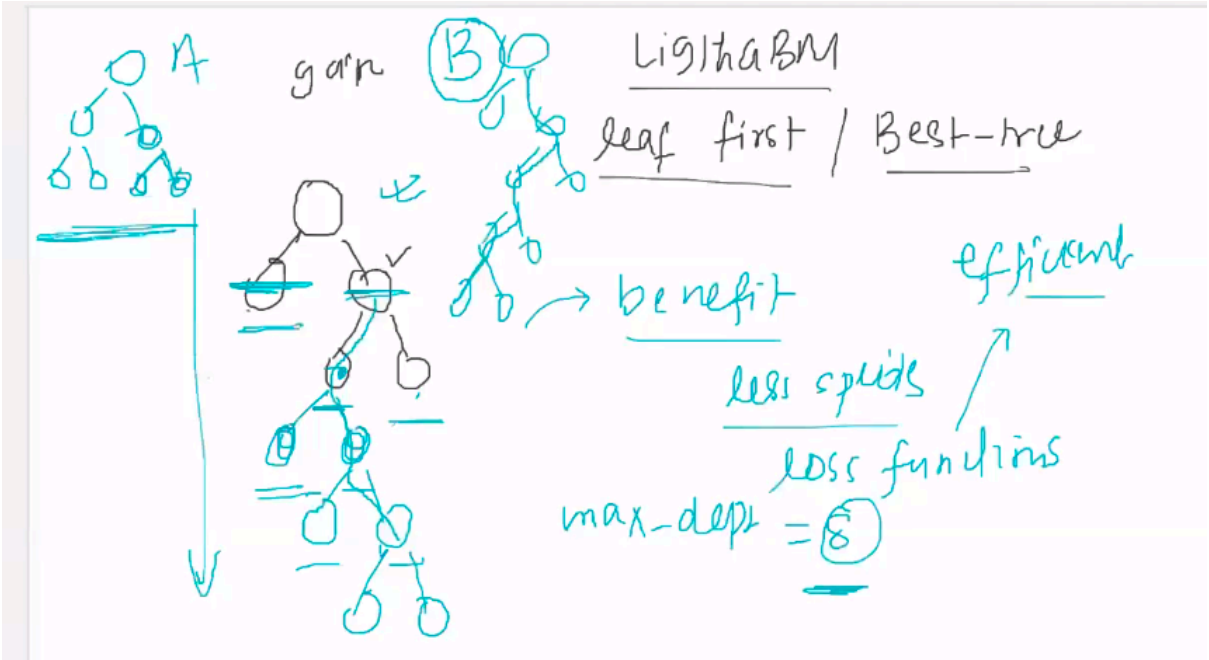

tru growing techm
GOSS
speed memory
Best fit tree
performance
EFB
speed memory


L
R
growing_techqm in decision


Growth Strategy
Depth first
Level first
breadth first
leaf-first
Best-tree


Depth-first
SK dT
GBDT
XGboost
max_depth
L
R
LL
LR
LLL
LLR
Y
N



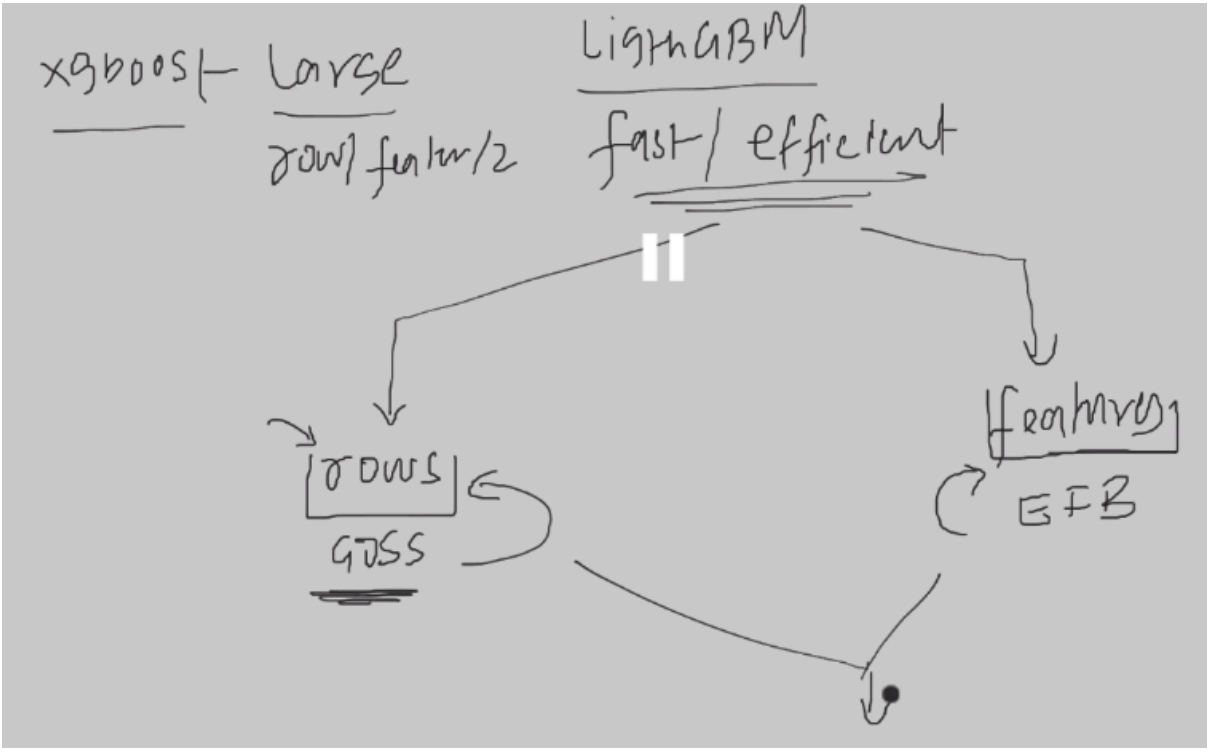


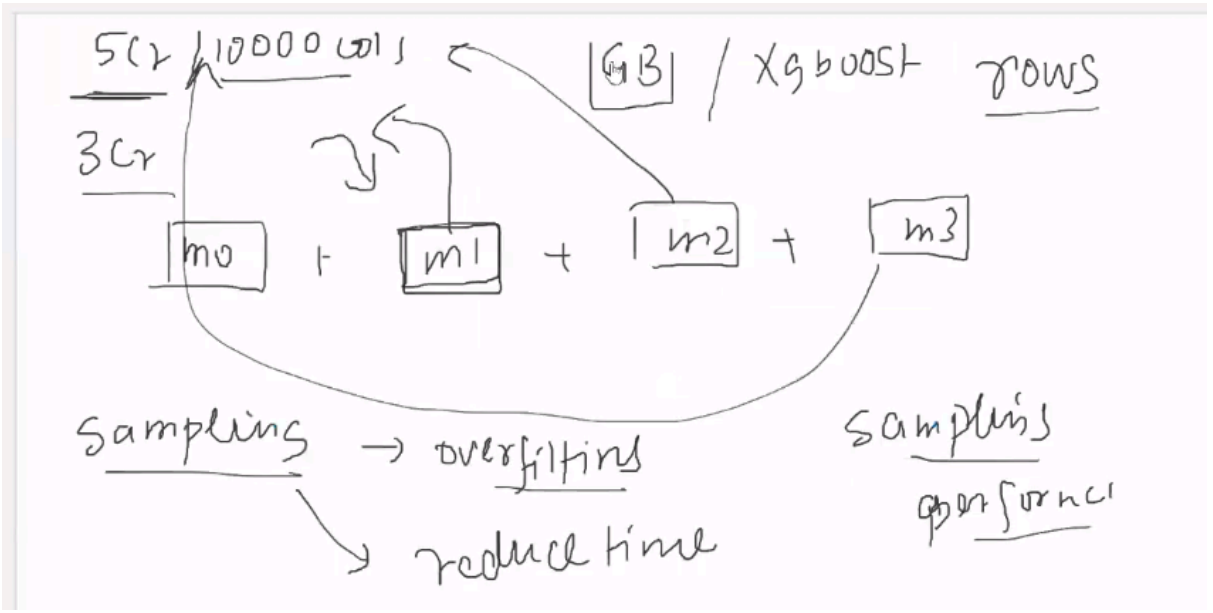
4. GOSS

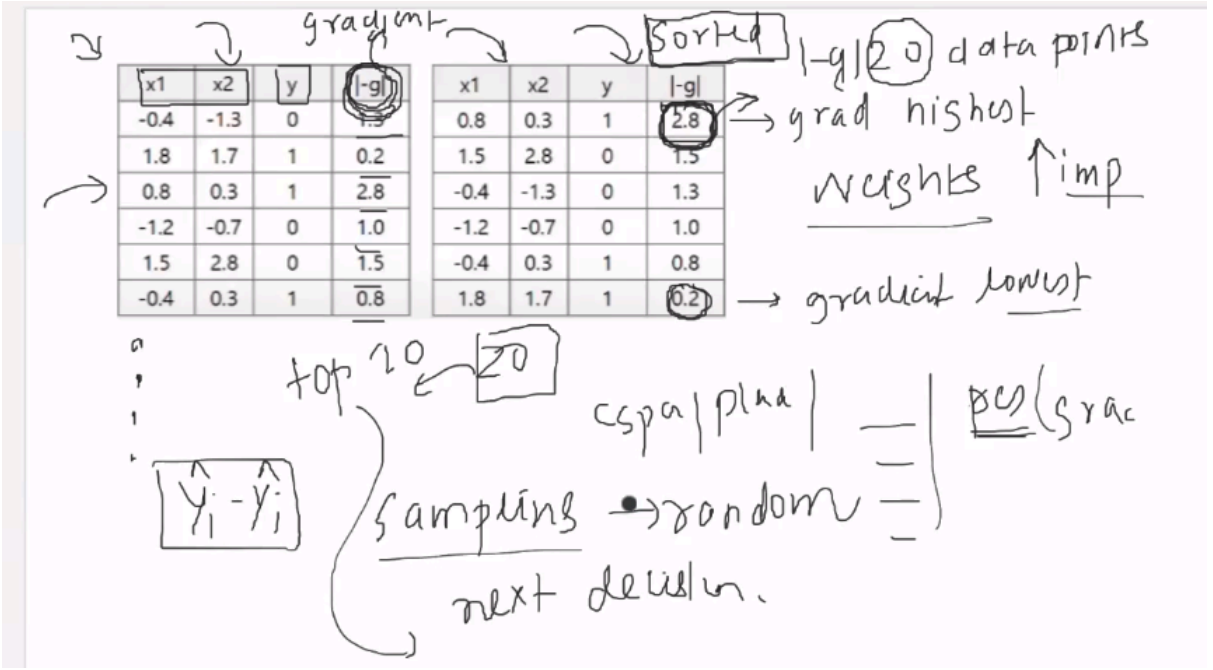
   GOSS is an innovative data sampling technique that addresses a common problem in the training of gradient boosting models: the balancing act between speeding up training by using a subset of the data (data sampling), and maintaining the accuracy of the learned model.

x1	x2	y	|-g|
-0.4	-1.3	0	1.3
1.8	1.7	1	0.2
0.8	0.3	1	2.8
-1.2	-0.7	0	1.0
1.5	2.8	0	1.5
-0.4	0.3	1	0.8

x1	x2	y	|-g|
0.8	0.3	1	2.8
1.5	2.8	0	1.5
-0.4	-1.3	0	1.3
-1.2	-0.7	0	1.0
-0.4	0.3	1	0.8
1.8	1.7	1	0.2

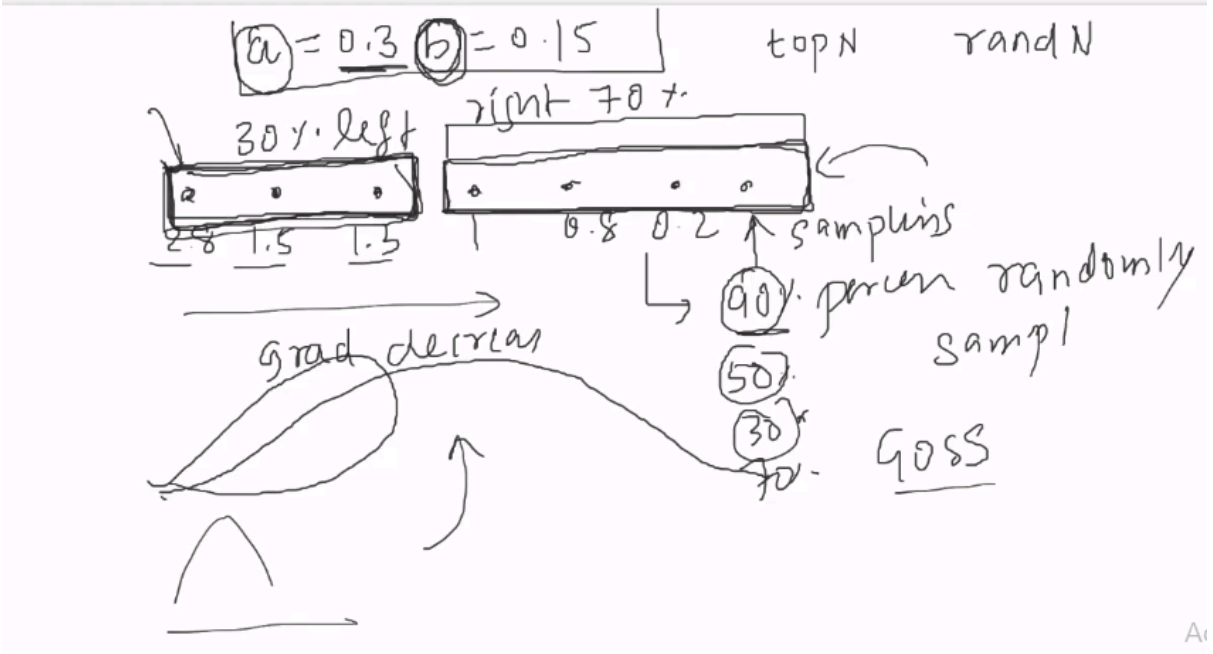


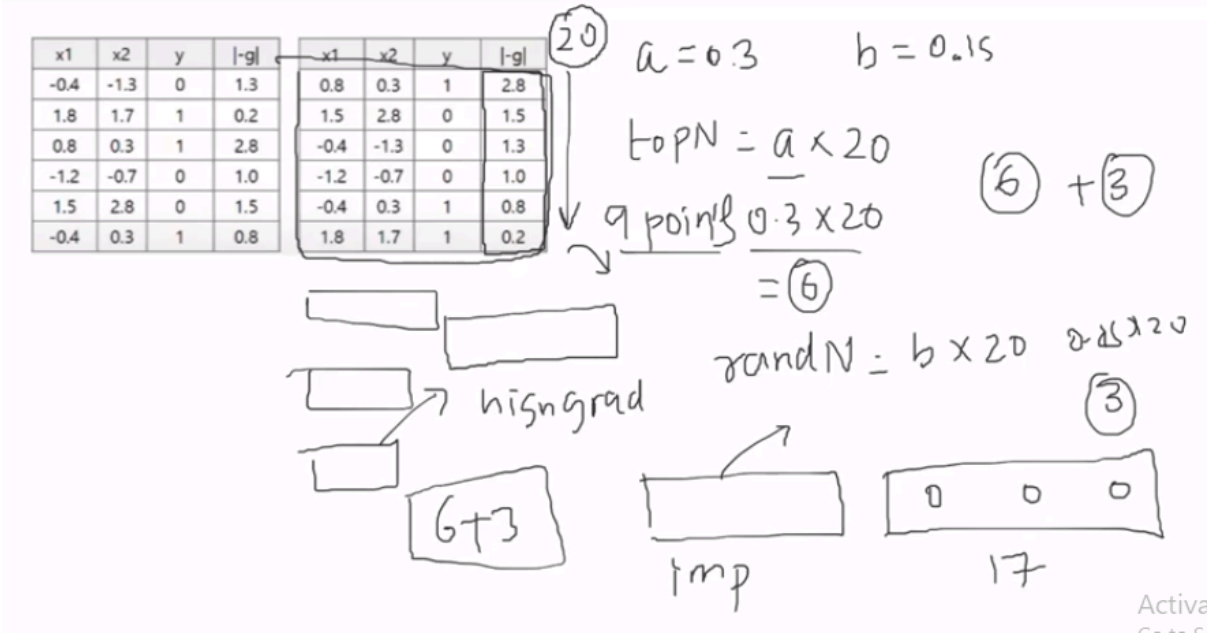




x1	x2	y	|-g|
-0.4	-1.3	0	1.5
1.8	1.7	1	0.2
0.8	0.3	1	2.8
-1.2	-0.7	0	1.0
1.5	2.8	0	1.5
-0.4	0.3	1	0.8

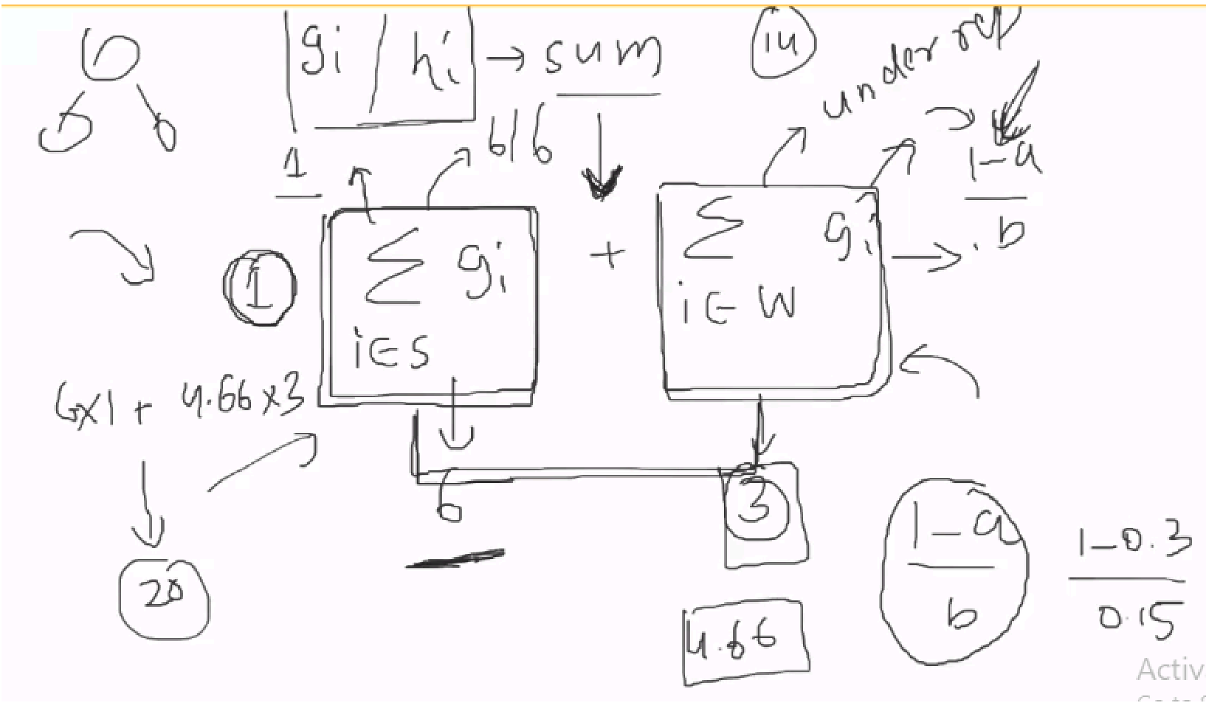
x1	x2	y	|-g|
0.8	0.3	1	2.8
1.5	2.8	0	1.5
-0.4	-1.3	0	1.3
-1.2	-0.7	0	1.0
-0.4	0.3	1	0.8
1.8	1.7	1	0.2





x1	x2	y	|-g|
-0.4	-1.3	0	1.3
1.8	1.7	1	0.2
0.8	0.3	1	2.8
-1.2	-0.7	0	1.0
1.5	2.8	0	1.5
-0.4	0.3	1	0.8

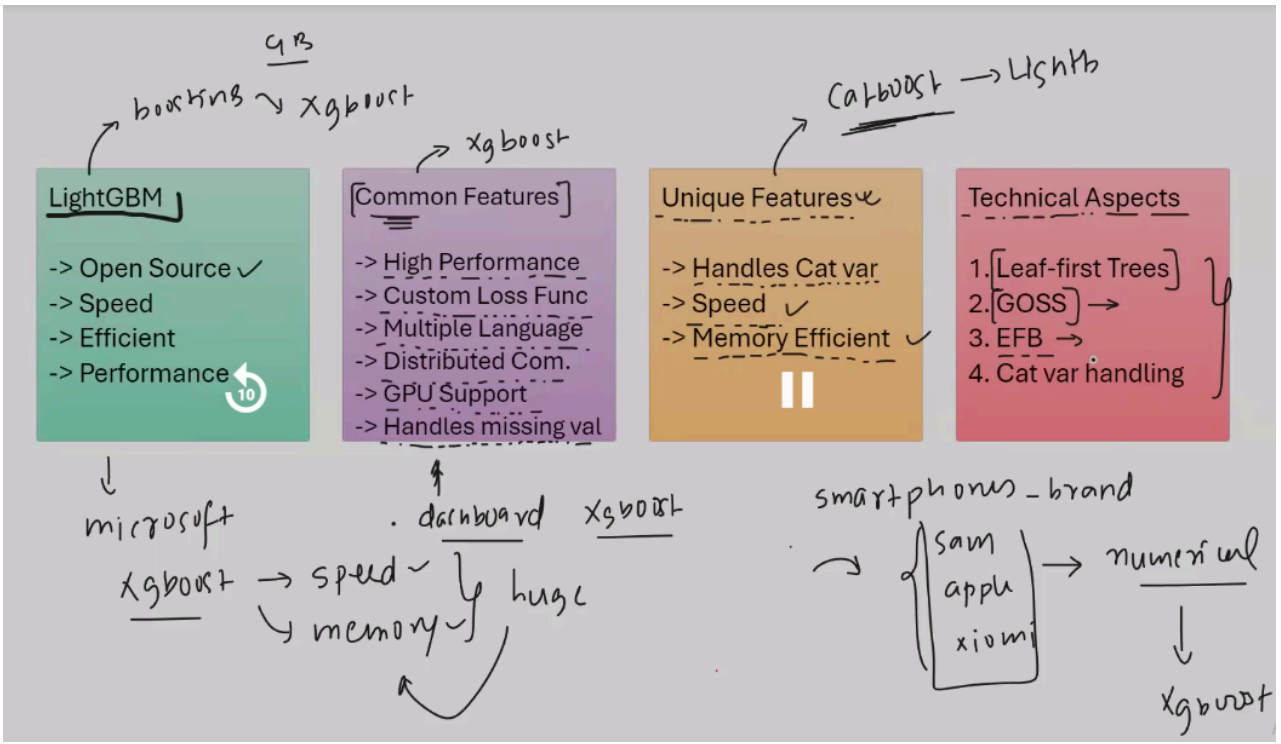
x1	x2	y	|-g|
0.8	0.3	1	2.8
1.5	2.8	0	1.5
-0.4	-1.3	0	1.3
-1.2	-0.7	0	1.0
-0.4	0.3	1	0.8
1.8	1.7	1	0.2

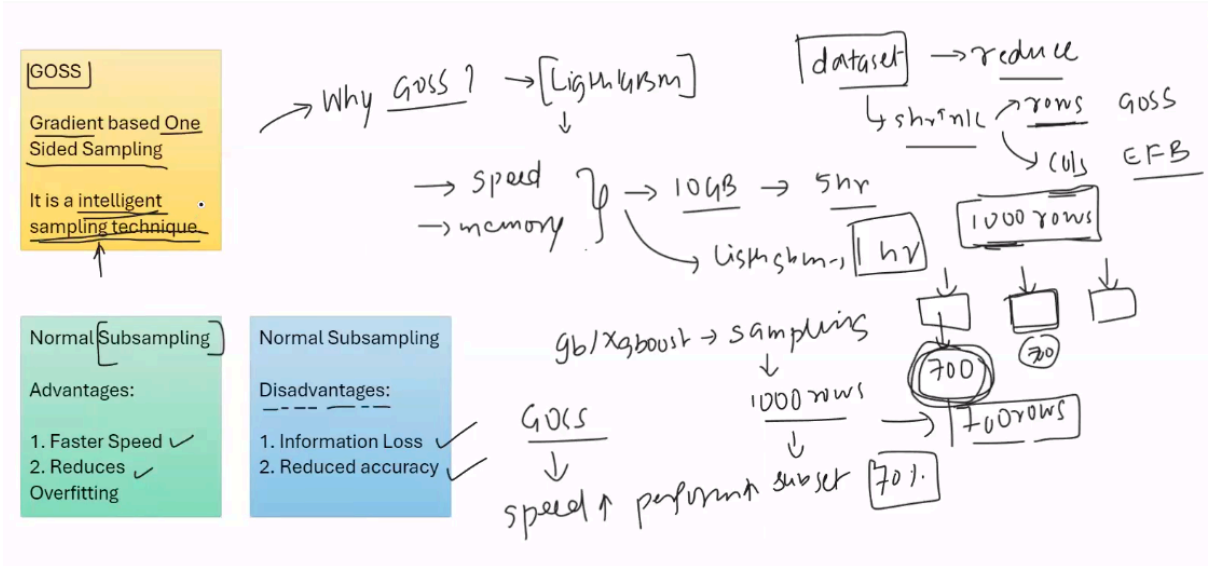


gi / hi → sum
14
under ro
1
6/6
Σ gi
i∈S
+
Σ gi
i∈W
1−a
b
6x1 + 4.66x3
20
6
3
4.66
1−a
b
1−0.3
0.15

Session - 2 LightGBM

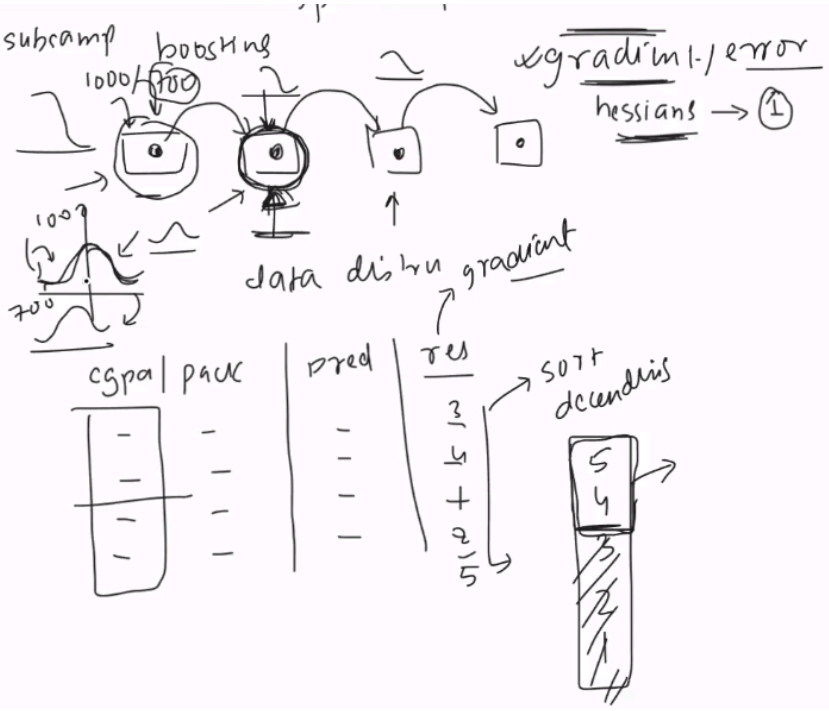
1.Recap





GOSS Main Idea

Some points are more important that other points!



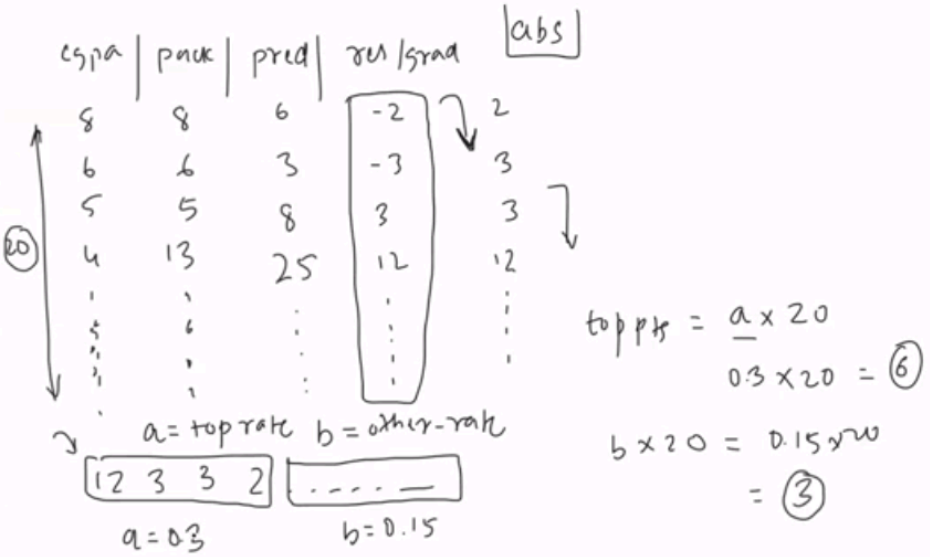
Simplified Solution

Send high gradient points to the next boosting round.

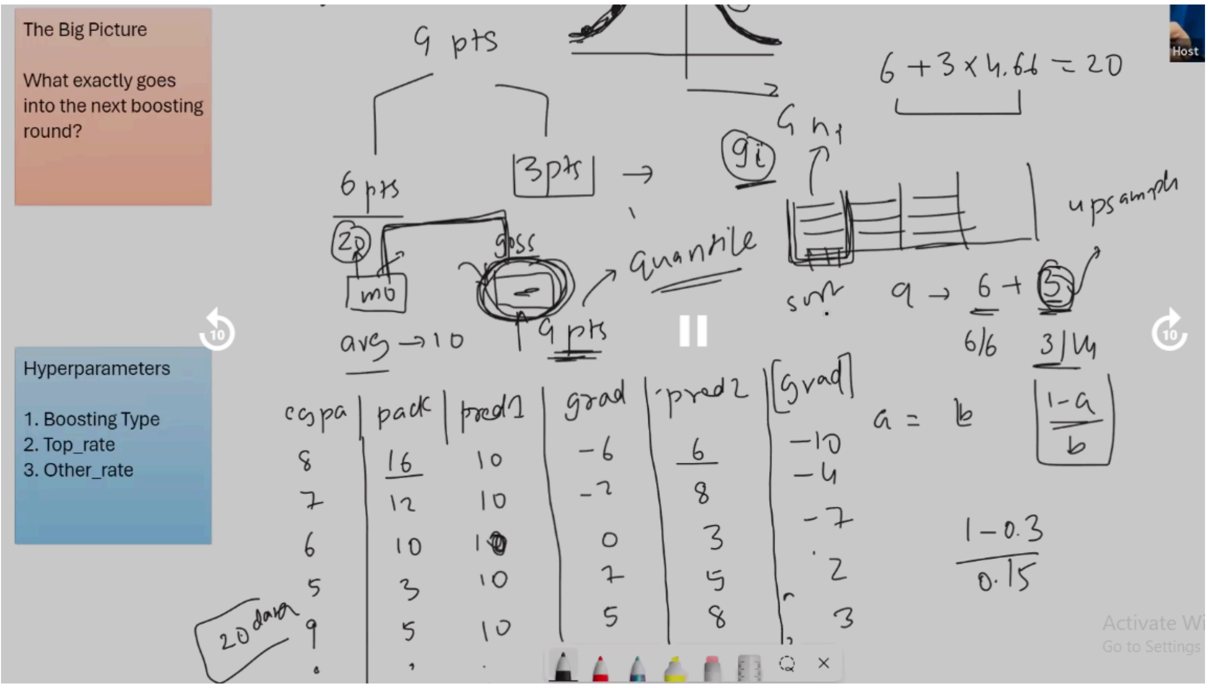
Problem with this

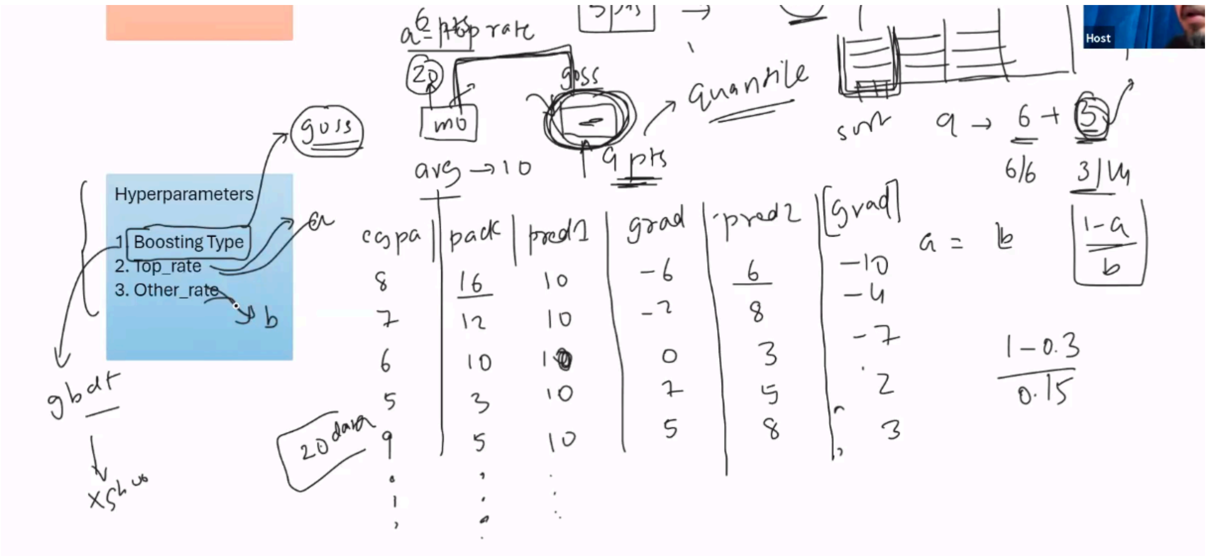
Refined Approach

Best of both worlds!



How Exactly?


The Big Picture
What exactly goes into the next boosting round?
Hyperparameters
1. Boosting Type
2. Top_rate
3. Other_rate
Host


Hyperparameters
1. Boosting Type
2. Top_rate
3. Other_rate
Host

3. EFB

3. EFB

Purpose of EFB:

Exclusive Feature Bundling is designed to reduce the dimensionality of the dataset by combining sparse and exclusive features into a single feature. This helps in managing datasets with a large number of features, where many of the features do not co-occur frequently.

Mechanism of Action:

EFB works by identifying features that are "exclusive," meaning their non-zero values rarely overlap with non-zero values of other features. These features are then combined into bundles where each original feature occupies a unique range of values, achieved by adding offsets.

Benefits for Machine Learning Models:

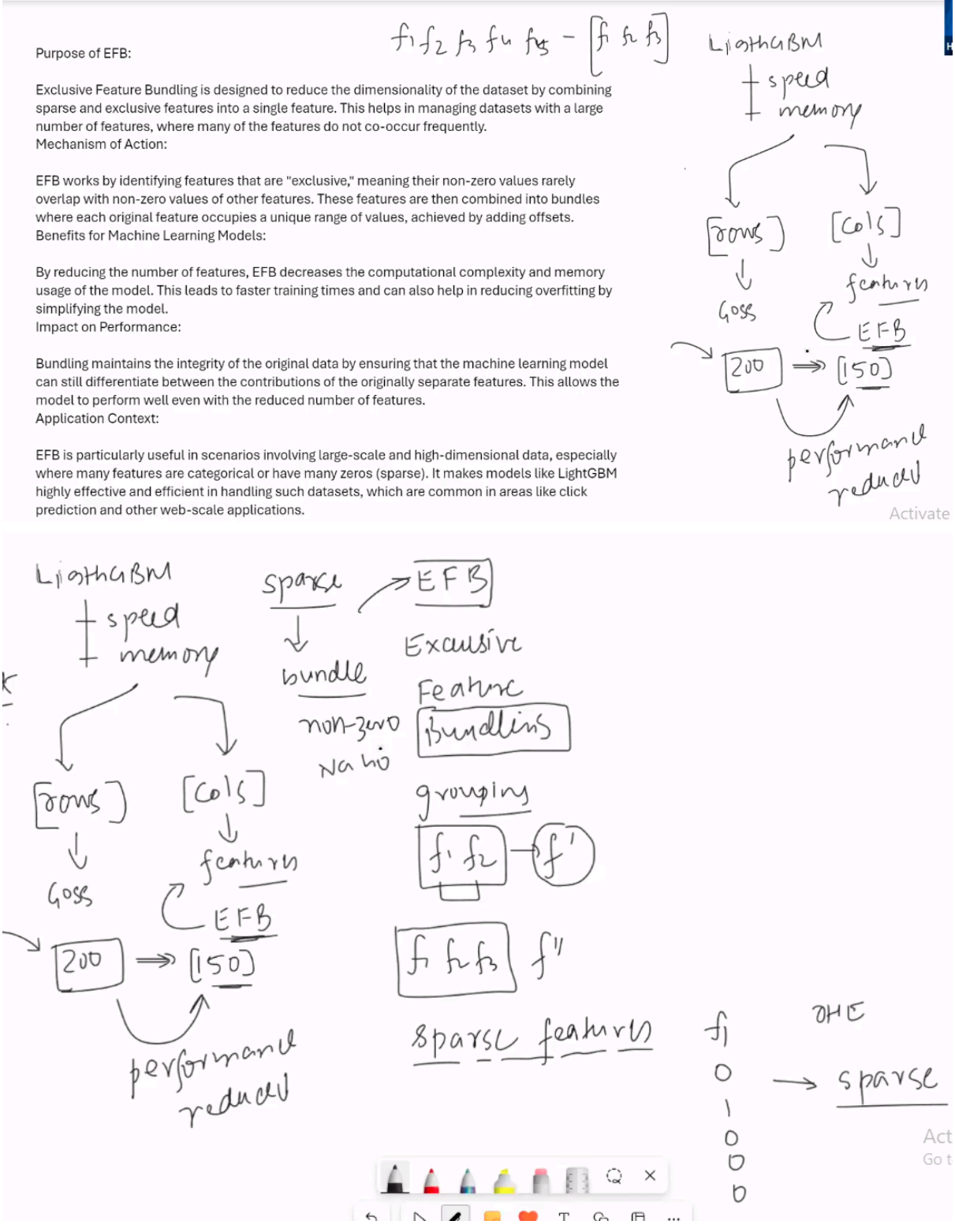
By reducing the number of features, EFB decreases the computational complexity and memory usage of the model. This leads to faster training times and can also help in reducing overfitting by simplifying the model.

Impact on Performance:

Bundling maintains the integrity of the original data by ensuring that the machine learning model can still differentiate between the contributions of the originally separate features. This allows the model to perform well even with the reduced number of features.

Application Context:

EFB is particularly useful in scenarios involving large-scale and high-dimensional data, especially where many features are categorical or have many zeros (sparse). It makes models like LightGBM highly effective and efficient in handling such datasets, which are common in areas like click prediction and other web-scale applications.



F0 F3 ③ 5 cols

F4 F2 Features

F1

	F0	F1	F2	F3	F4
i0	1	1	0	0	1
i1	0	0	1	1	1
i2	1	2	0	0	2
i3	0	0	2	3	1
i4	2	1	0	0	3
i5	3	3	0	0	1
i6	0	0	3	0	2
i7	1	2	3	4	3
i8	1	0	1	0	0
i9	2	3	0	0	2

EFB

1) find out cols that can be bundled together } Greedy Bundling

2) How to bundle } merge exclusive features

[greedy bundle] ✓ [MEf] ✓

SE

F0 F3 ③ 5 cols

F4 F2 Features

F1

	F0	F1	F2	F3	F4
i0	1	1	0	0	1
i1	0	0	1	1	1
i2	1	2	0	0	2
i3	0	0	2	3	1
i4	2	1	0	0	3
i5	3	3	0	0	1
i6	0	0	3	0	2
i7	1	2	3	4	3
i8	1	0	1	0	0
i9	2	3	0	0	2

conflict count matrix

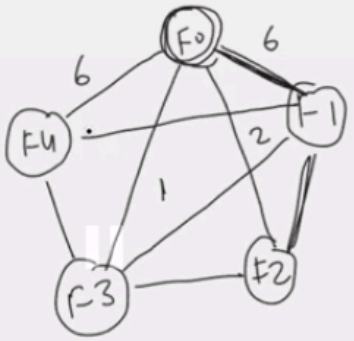
	F0	F1	F2	F3	F4
F0		6	2	1	
F1	6				
F2					
F3					
F4					

conflict count matrix

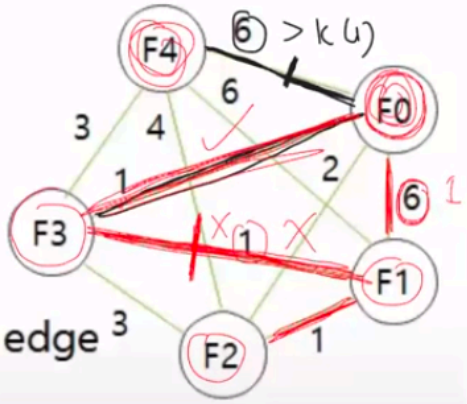
	F0	F1	F2	F3	F4
F0		6	2	1	6
F1	6		1	1	6
F2	2	1		3	4
F3	1	1	3		3
F4	6	6	4	3	
∑	15	14	10	8	19

graph ↗ data structure

nodes



graph G



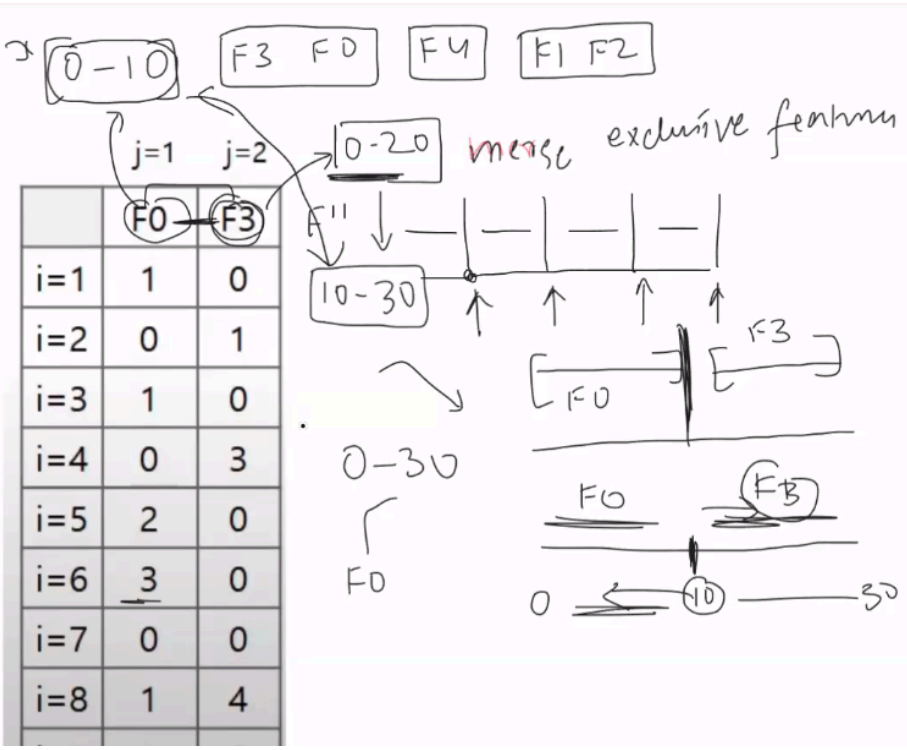
max conflict (k) = 1

F3 F2 1 X → 5 → 3

F3 F0 → f''

F2 F1 → f''''

F4 → f4



	F0	F3
i=1	1	0
i=2	0	1
i=3	1	0
i=4	0	3
i=5	2	0
i=6	3	0
i=7	0	0
i=8	1	4

Algorithm 4: Me

Input: *numData*:
Input: F: One bun
binRanges ← {0}, t
for *f in F* **do**
  totalBin += f.r
  binRanges.appe
newBin ← new Bi
for $i = 1$ *to numD*
  newBin[i] ← 0
  for $j = 1$ *to* le
    if *F[j].bin*
      newBi

Output: *newBin*,

3. EFB

F = {F0, F3} binRanges = [0, 3, 7] totalBins = 0

1 + 0

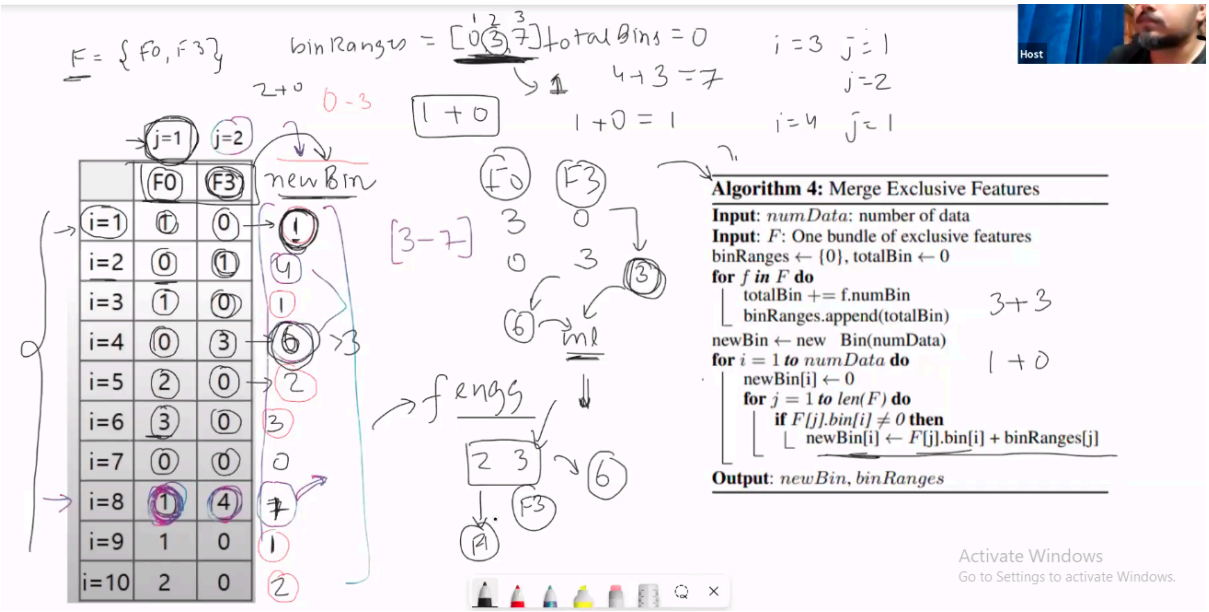
i=1 j=1
i=1 j=2
i=2 j=1
i=2 j=2

	F0	F3	newBin
i=1	1	0	1
i=2	0	1	0
i=3	1	0	
i=4	0	3	
i=5	2	0	
i=6	3	0	
i=7	0	0	
i=8	1	4	
i=9	1	0	
i=10	2	0	

Algorithm 4: Merge Exclusive Features

Input: *numData*: number of data
Input: *F*: One bundle of exclusive features
binRanges ← {0}, totalBin ← 0
for *f in F* **do**
  totalBin += f.numBin
  binRanges.append(totalBin)
newBin ← new Bin(numData)
for $i = 1$ **to** *numData* **do**
  newBin[i] ← 0
  for $j = 1$ **to** *len(F)* **do**
    if *F[j].bin[i]* ≠ 0 **then**
      newBin[i] ← F[j].bin[i] + binRanges[j]

Output: *newBin*, *binRanges*


F = { F0, F3 }
binRanges = [0 3 7] totalBins = 0
i=3 j=1
j=2
4+3=7
1+0
1+0=1
i=4 j=1
0-3
newBin
j=1
j=2
F0
F3
i=1 1 0
i=2 0 1
i=3 1 0
i=4 0 3
i=5 2 0
i=6 3 0
i=7 0 0
i=8 1 4
i=9 1 0
i=10 2 0
[3-7]
fengs
2 3
F3
A
Algorithm 4: Merge Exclusive Features
Input: numData: number of data
Input: F: One bundle of exclusive features
binRanges ← {0}, totalBin ← 0
for f in F do
totalBin += f.numBin
binRanges.append(totalBin)
newBin ← new Bin(numData)
for i = 1 to numData do
newBin[i] ← 0
for j = 1 to len(F) do
if F[j].bin[i] ≠ 0 then
newBin[i] ← F[j].bin[i] + binRanges[j]
Output: newBin, binRanges
3+3
1+0
Activate Windows
Go to Settings to activate Windows.