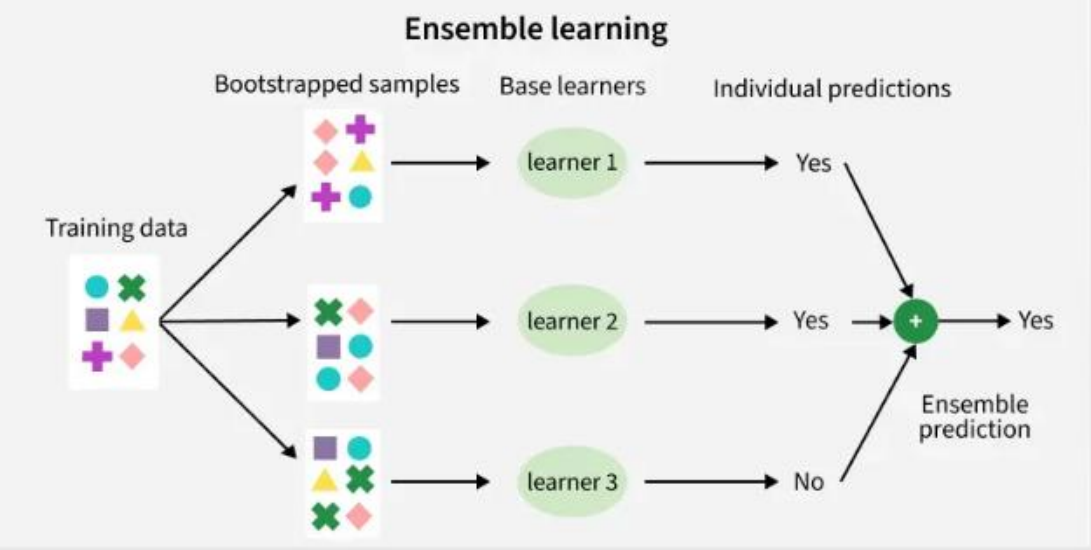
Boosting Algorithm - Regression

MACHINE LEARNING ALGORITHM

Ensemble Learning

Ensemble learning is a method where we **use many small models instead of just one**. Each of these models may not be very strong on its own, but when we put their results **together, we get a better and more accurate answer.**

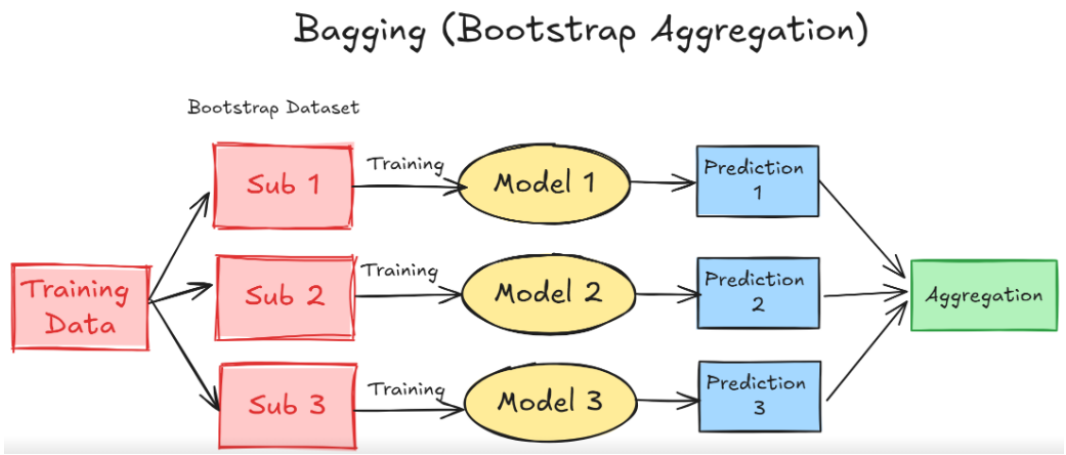


Ensemble Learning = **combining multiple models** to get **better accuracy** than a single model.

Ensemble Learning Types

1. Bagging
2. Boosting
3. Stacking

Bagging



As you can see in the process, the training data becomes a subset of the original via bootstrap.

Original Data

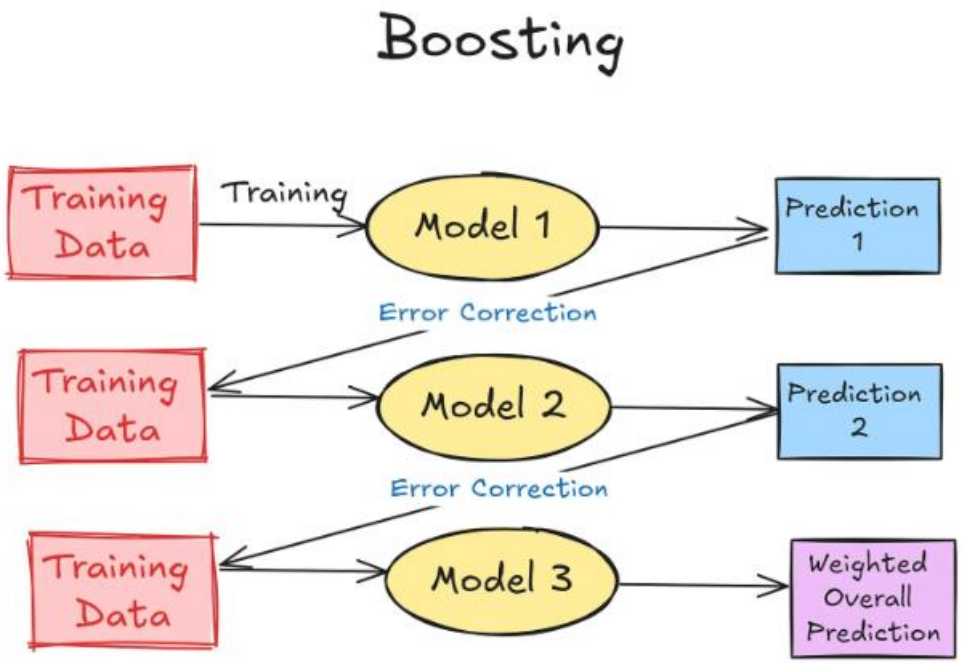
Size	Color	Shape	Taste	Rebuy
L	Red	Square	Good	Yes
M	Blue	Circle	Bad	Yes
S	Red	Triangle	Good	No

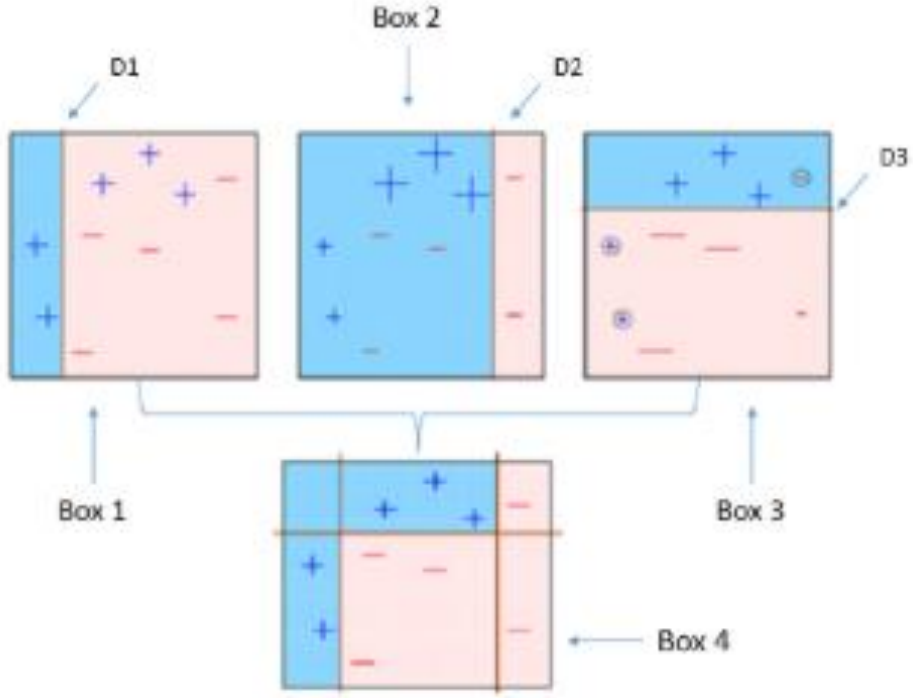
Bootstrap Dataset

Size	Color	Shape	Taste	Rebuy
L	Red	Square	Good	Yes
S	Red	Triangle	Good	No
S	Red	Triangle	Good	No
L	Red	Square	Good	Yes

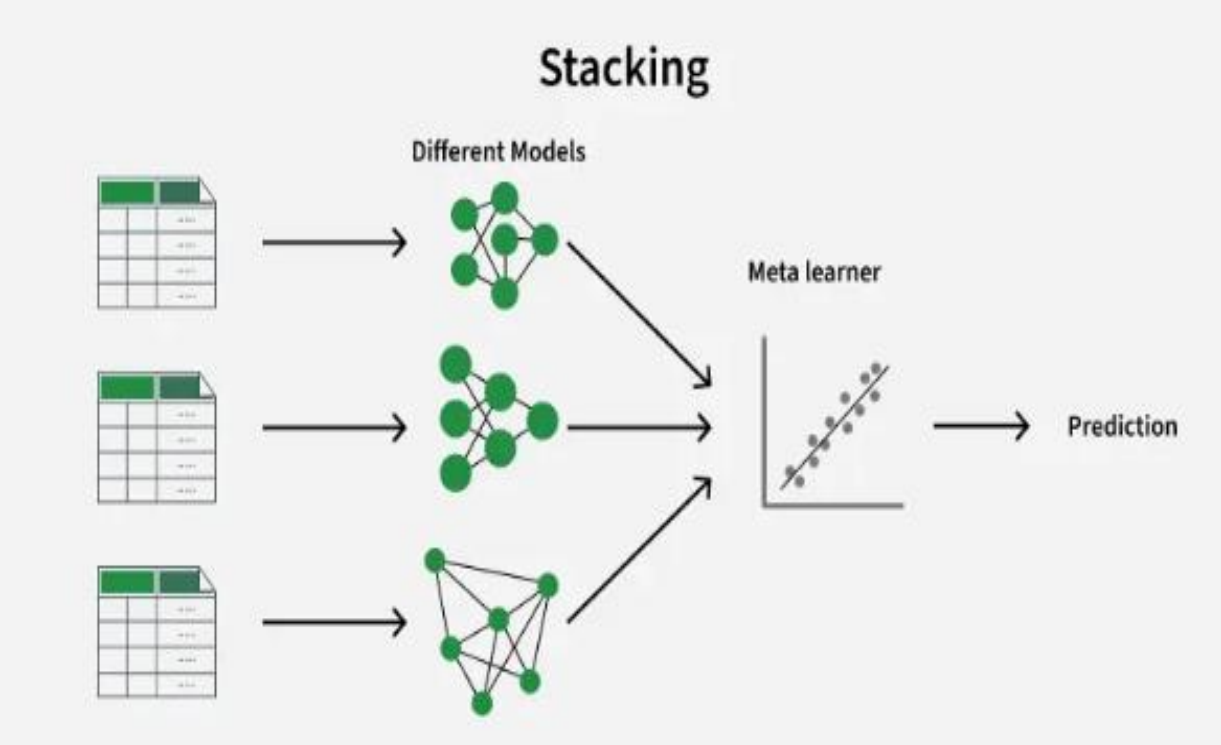
The table above shows how Bootstrap. The same data can be shown multiple times, and some data points are not even present in the Bootstrap dataset.

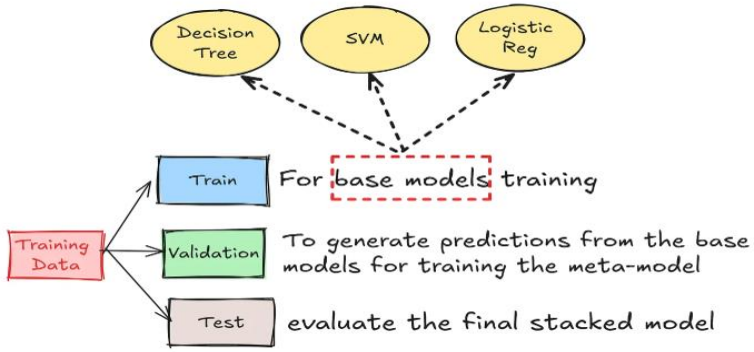
Boosting


Boosting
Training Data
Training
Model 1
Prediction 1
Error Correction
Training Data
Model 2
Prediction 2
Error Correction
Training Data
Model 3
Weighted Overall Prediction


Box 2
D1
D2
D3
Box 1
Box 3
Box 4

Stacking





EXAMPLE:
Imagine 3 doctors checking a patient:
•Doctor 1: heart specialist
•Doctor 2: lung specialist
•Doctor 3: general physician
All give opinions → Head doctor (meta-model) takes all opinions → final decision.
This is exactly how Stacking works.

Over fitting and Under fitting

Under Fitting :

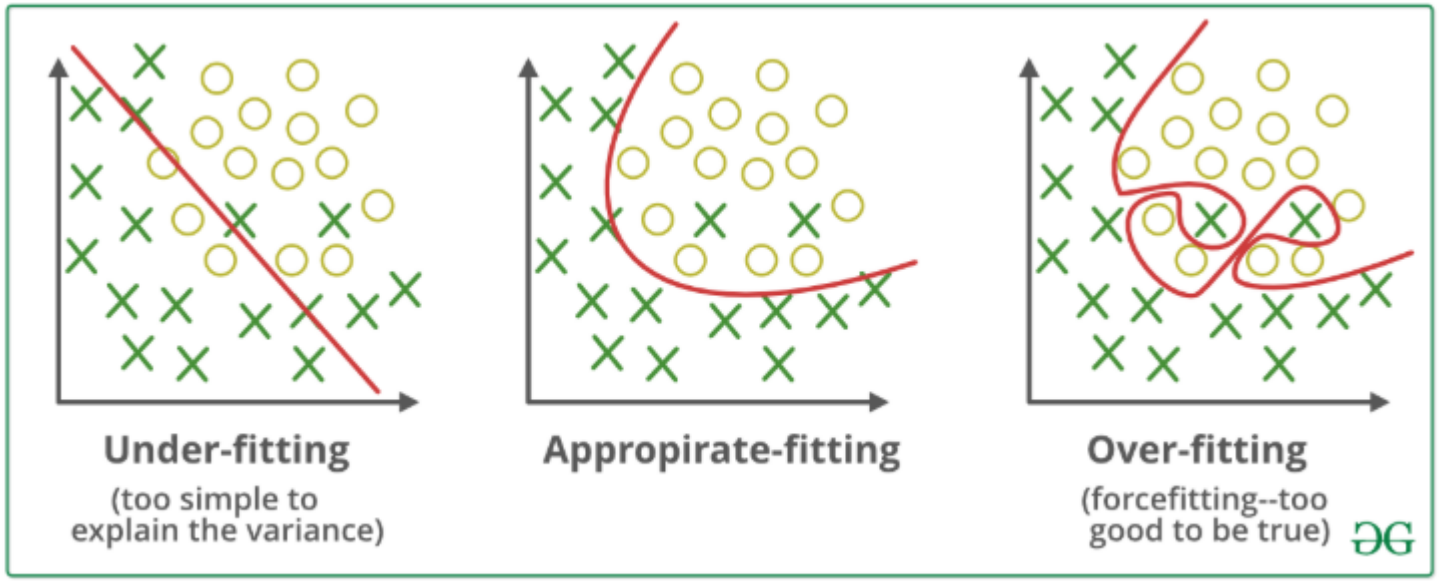
- model is too simple to capture the underlying patterns in the training data
- **High bias, high error rate**
- High error on both train and test set
- Poor model performance**, model is too simple**

Over Fitting :

- model learns the training data too well, including its noise, and fails to generalize to new, unseen data
- **High Variance**
- low error on training data and high error on test data. (**Memorization, not learning)**
- **Model complexity**

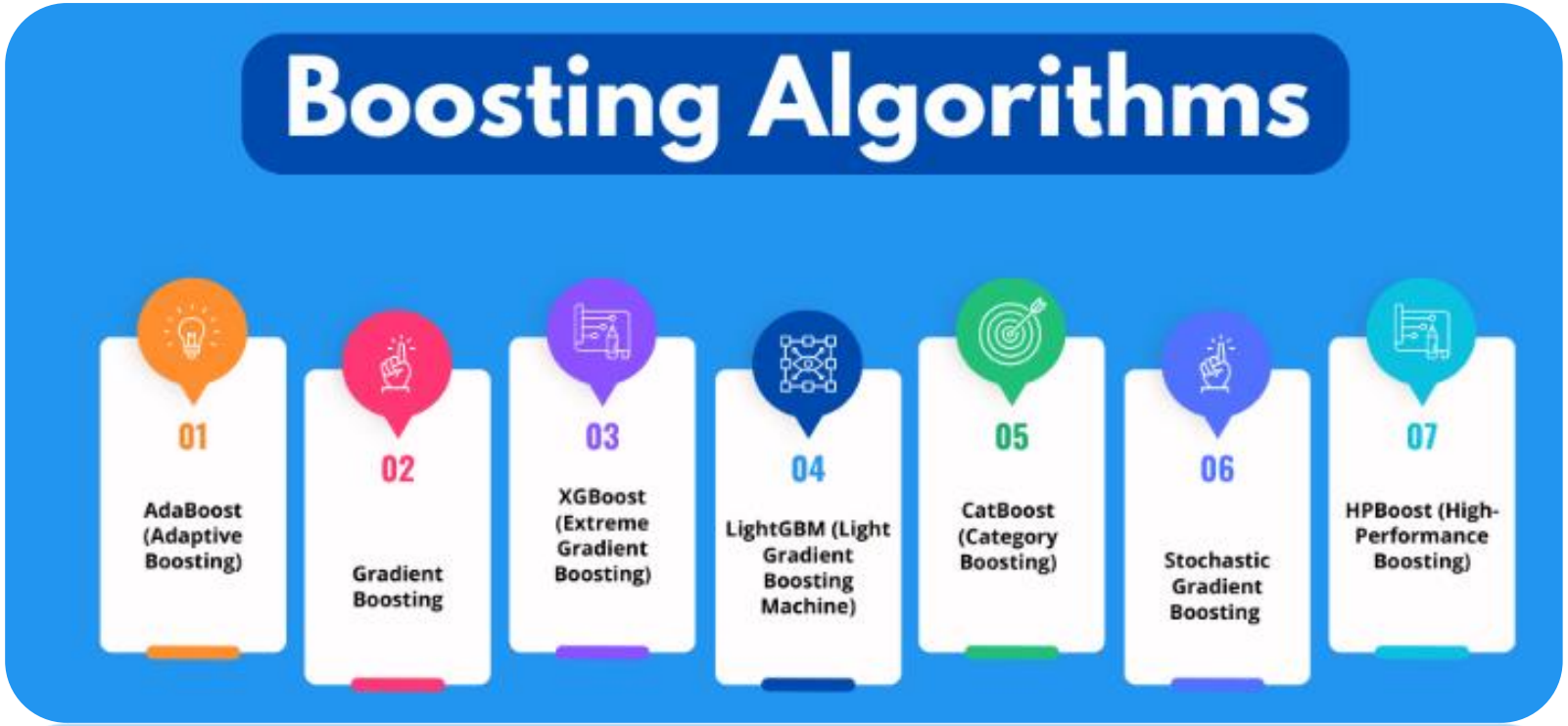
Appropriate Fitting :

- **good fit or best fit**
- model that effectively captures the underlying trends and patterns in the training data without being overly complex or simplistic
- It represents a balance between under fitting and over fitting.



Boosting Algorithm Types

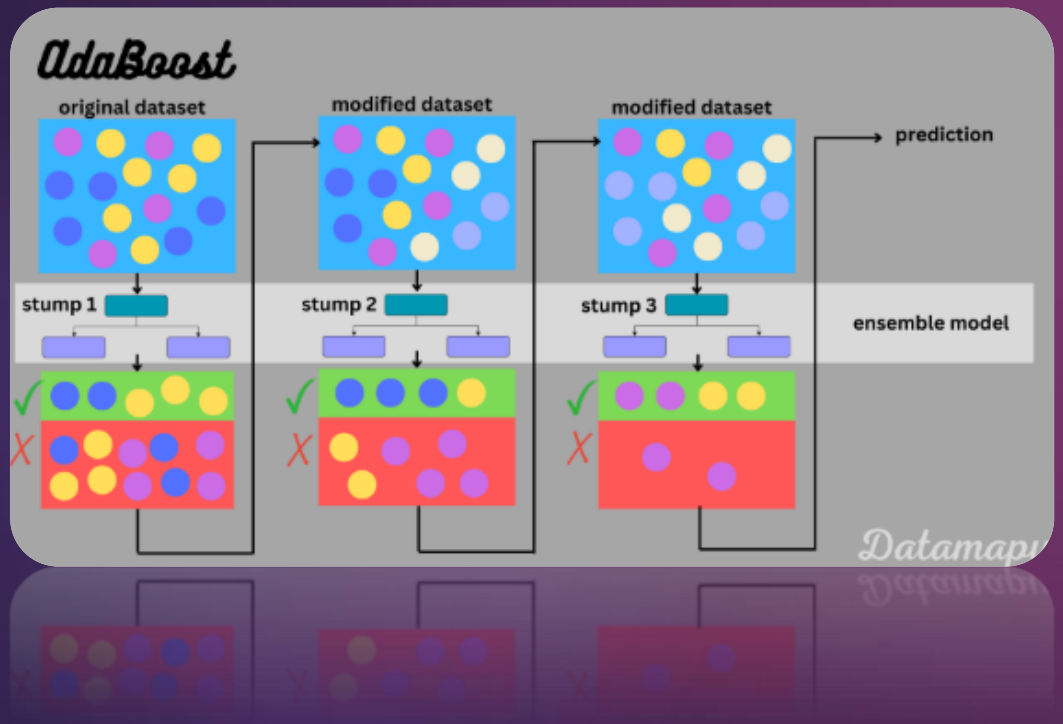
- **Ada Boost** (Adaptive Boosting)
- **XG boost** (Extreme gradient boosting)
- **LG boost** (light GBM – light gradient boosting machine)



Ada Boost algorithm

- Same as normal boosting algorithm
- AdaBoost algorithm, short for **Adaptive Boosting**, is a Boosting technique used as an Ensemble Method in Machine Learning.
- It is called Adaptive Boosting as the **weights are re-assigned to each instance, with higher weights assigned to incorrectly classified instances**.
- In simple words, weak learners are converted into strong ones. By Giving **Incremental weight concept** (weight-assigning technique)
- It can be used for **both Classification and Regression problems**. In **most cases**, it is used for **classification problems**.
- One can first try ***decision trees*** and then go for the ***random forest*** to finally apply the boost and implement ***AdaBoost***. **Accuracy keeps increasing as we follow the above sequence**.

Working of Ada Boost algorithm



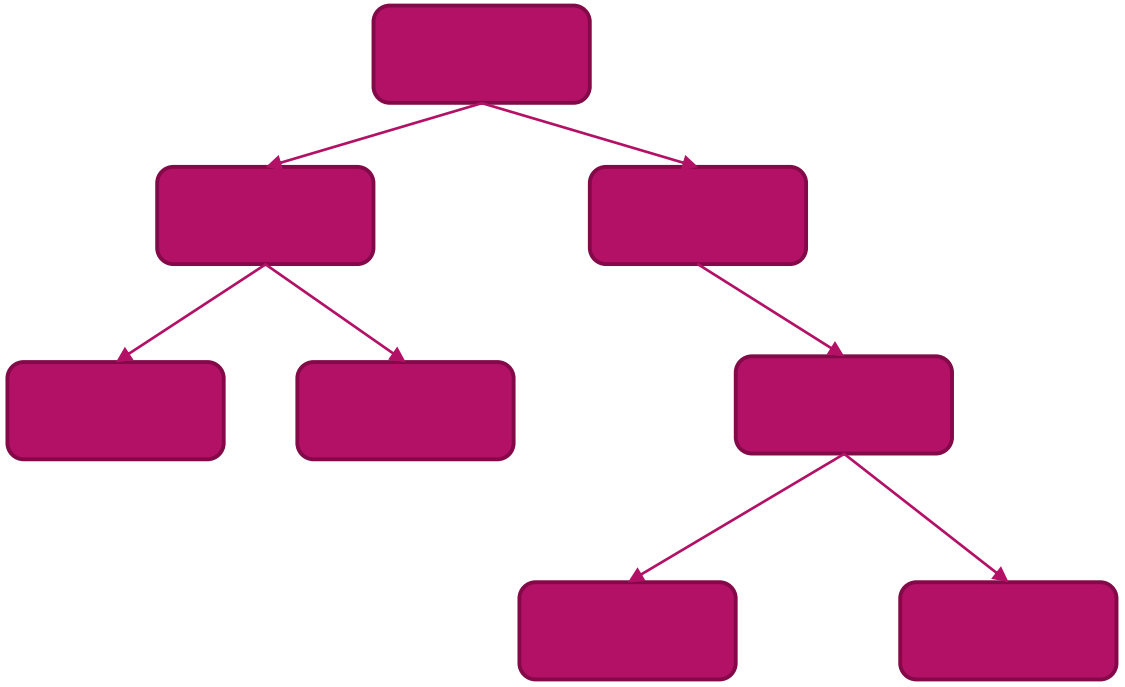
- Ada boost, Sample weight :

'**W=1/N**' where **N** is the number of records.

- Step 1: Creating base learner
- Step 2: Calculate the total error
- Step 3: Calculate the performance of the stump
- Step 4: Update the weight
- Step 5: Update the values in the new dataset
- How Does the Algorithm Decide Output for Test Data?
- How to Code AdaBoost in Python?

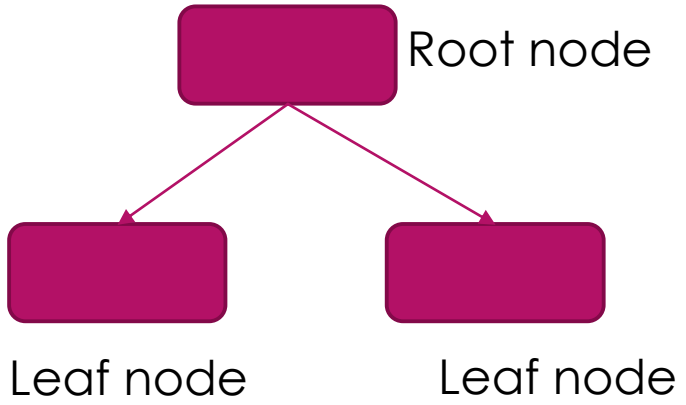
Random Forest Vs. Ada Boost

RANDOM FOREST

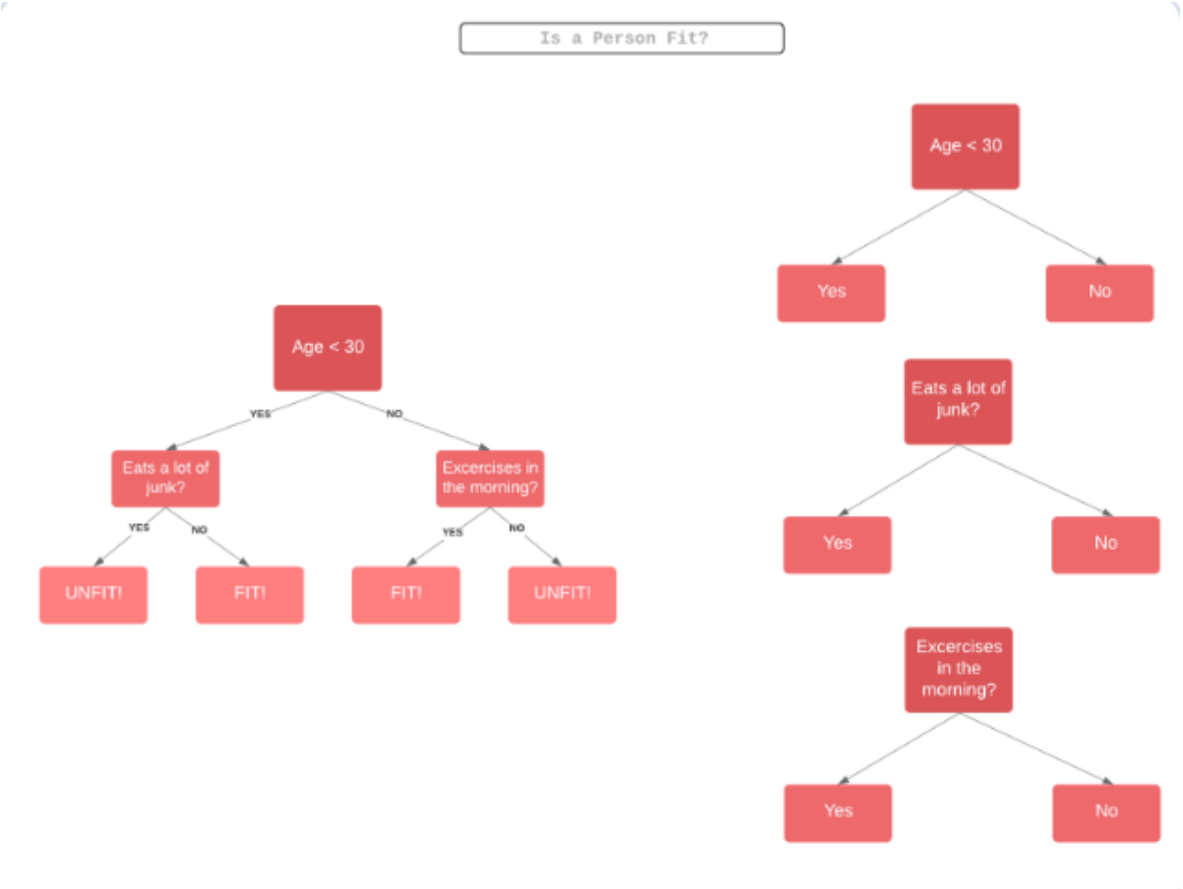


Tree

ADA BOOST



Stump



These stumps are weak learners and boosting techniques prefer this. The **order of stumps is very important in AdaBoost.** The error of the first stump influences how other stumps are made.

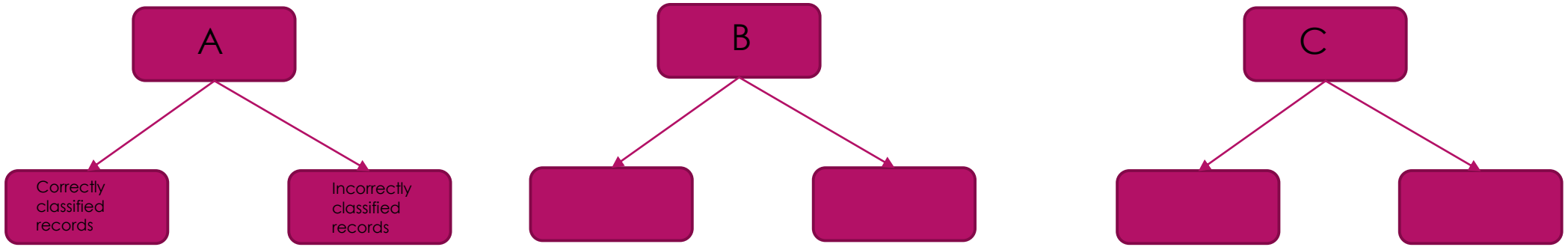
dataset

Row no	A	B	C	O/P	Sample weight
1	Categorical Data			Y	1/5
2				Y	1/5
3				N	1/5
4				N	1/5
5				Y	1/5

$w = 1/N$

$w = 1/5$

Step 1 – Creating the First Base Learner

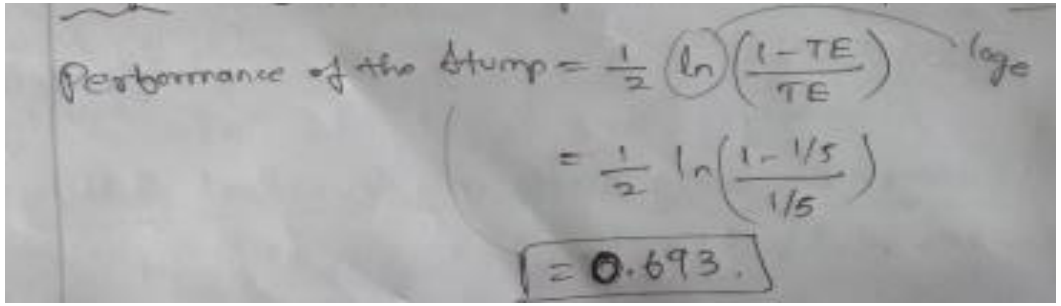


Out of these 3 models, the algorithm selects only one. Two properties are considered while selecting a base learner - **Gini and Entropy**. We must calculate Gini or Entropy the same way it is calculated for decision trees. The stump with the **least value** will be the first base learner.

Step 2 – Calculating the Total Error (TE)

Total Error (TE) = 1/5.

Step 3 – Calculating Performance of the Stump



We must increase the weight for the wrongly classified records and decrease the weight for the correctly classified records.

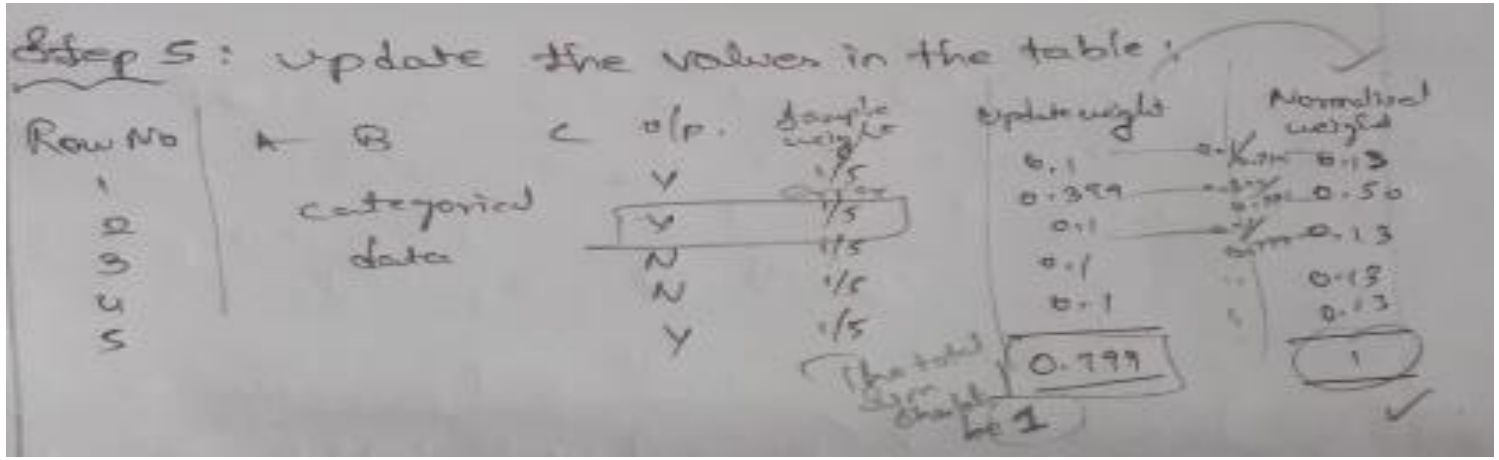
Step 4 – Updating Weights

For incorrectly classified records, the formula for updating weights is:
New Sample Weight = Sample Weight * e^(Performance)
In our case Sample weight = 1/5 so, **1/5 * e^ (0.693) = 0.399**

For correctly classified records, the formula is:
New Sample Weight = Sample Weight * e^- (Performance)
Putting the values, **1/5 * e^-(0.693) = 0.100**

Note: the total sum of all the weights should be 1 By ***normalized weight***.

Step 5 – Creating a New Dataset



How Does the Algorithm Decide Output for Test Data?

- The test dataset will pass through all the stumps which have been constructed by the algorithm.
- While passing through the **1st stump**, the output it produces is **1**. Passing through the **2nd stump**, the output generated once again is **1**. While passing through the **3rd stump** it gives the output as **0**.
- In the AdaBoost algorithm too, the **majority of votes** take place between the stumps, in the same way as in random trees.
- In this case, **the final output will be 1.**

1. In the new dataset, the **frequency of incorrectly classified records will be more** than the correct ones.
2. To make a new dataset based on **normalized weight**, the algorithm will **divide it into buckets.**
3. So, our first bucket is from **0 – 0.13,** second will be from **0.13 – 0.63(0.13+0.50),** third will be from **0.63 – 0.76(0.63+0.13),** and so on.
4. random value **0.46**
5. There is a **high probability for wrong records to get selected several times**.
6. Based on this new dataset, the algorithm will create a new decision tree/stump and it will **repeat the same process from step 1 till it sequentially passes through all stumps and finds that there is less error** as compared to normalized weight

https://www.mygreatlearning.com/blog/adaboost-algorithm/

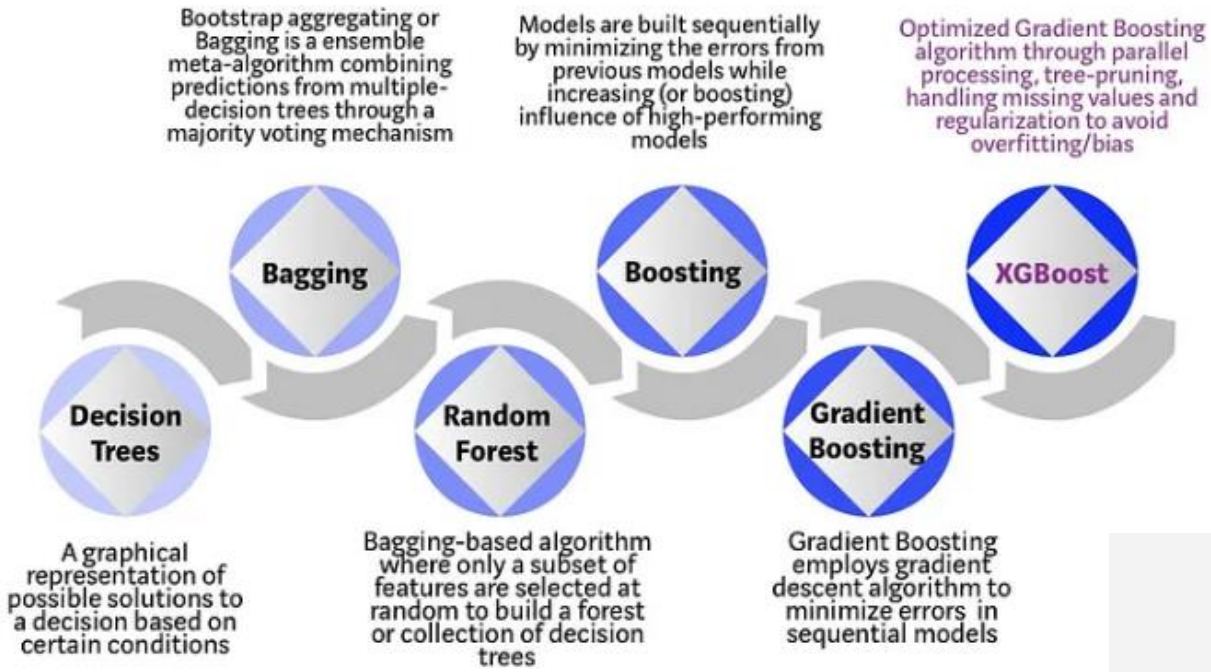
Ada Boost Coding in Python :

My CODE :

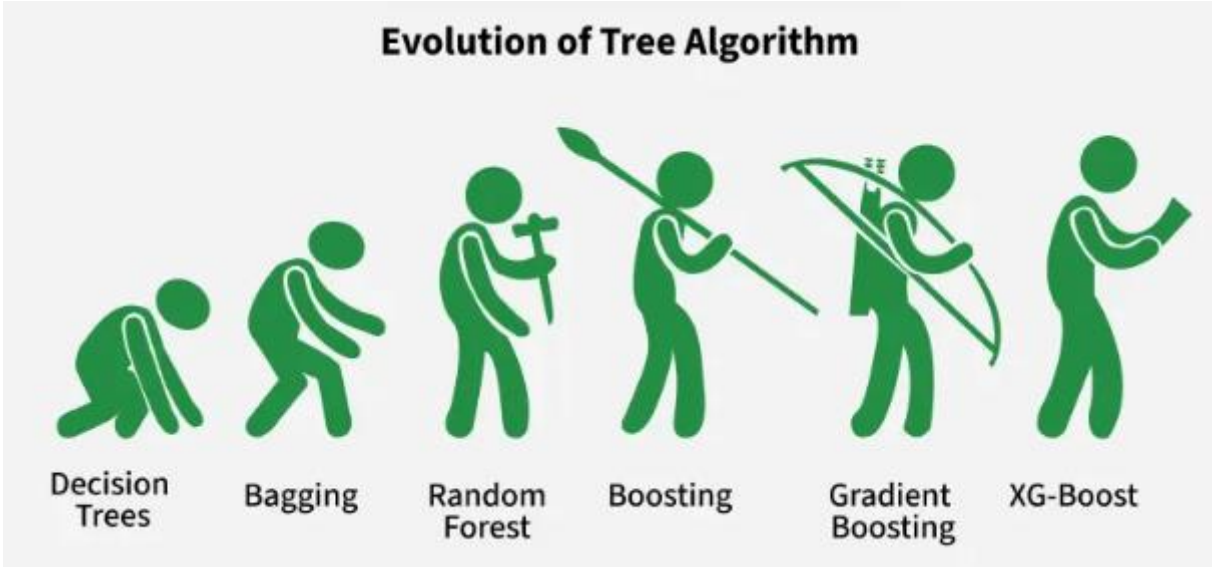
https://github.com/krthiksha/Machine-Learning-Regression_module_HopeAI/blob/main/Boosting%20Algorithm%20-%20Regression/AdaBoost_insurance_charge_prediction.ipynb

Documentation :

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html



Evolution of XGBoost Algorithm from Decision Trees

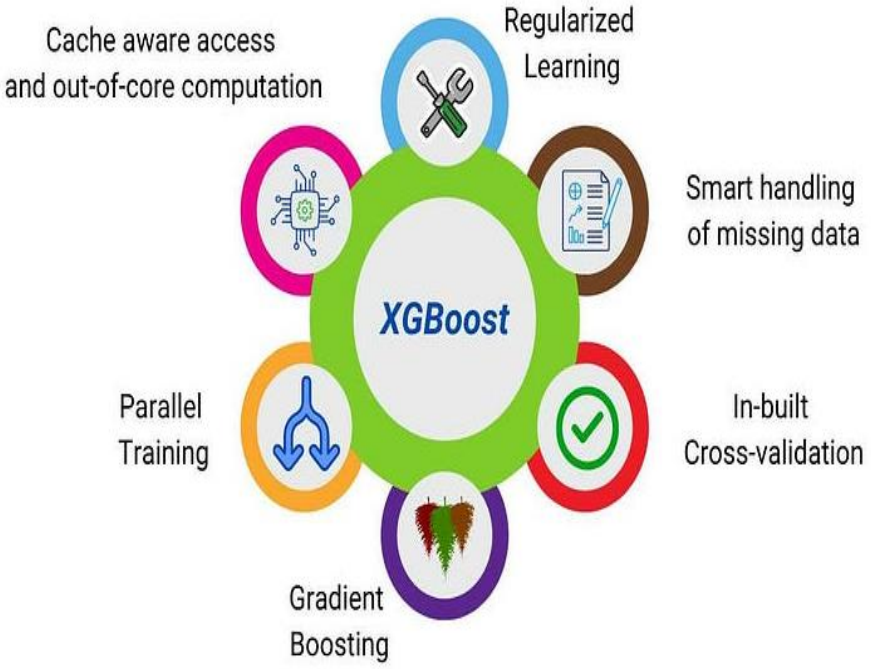


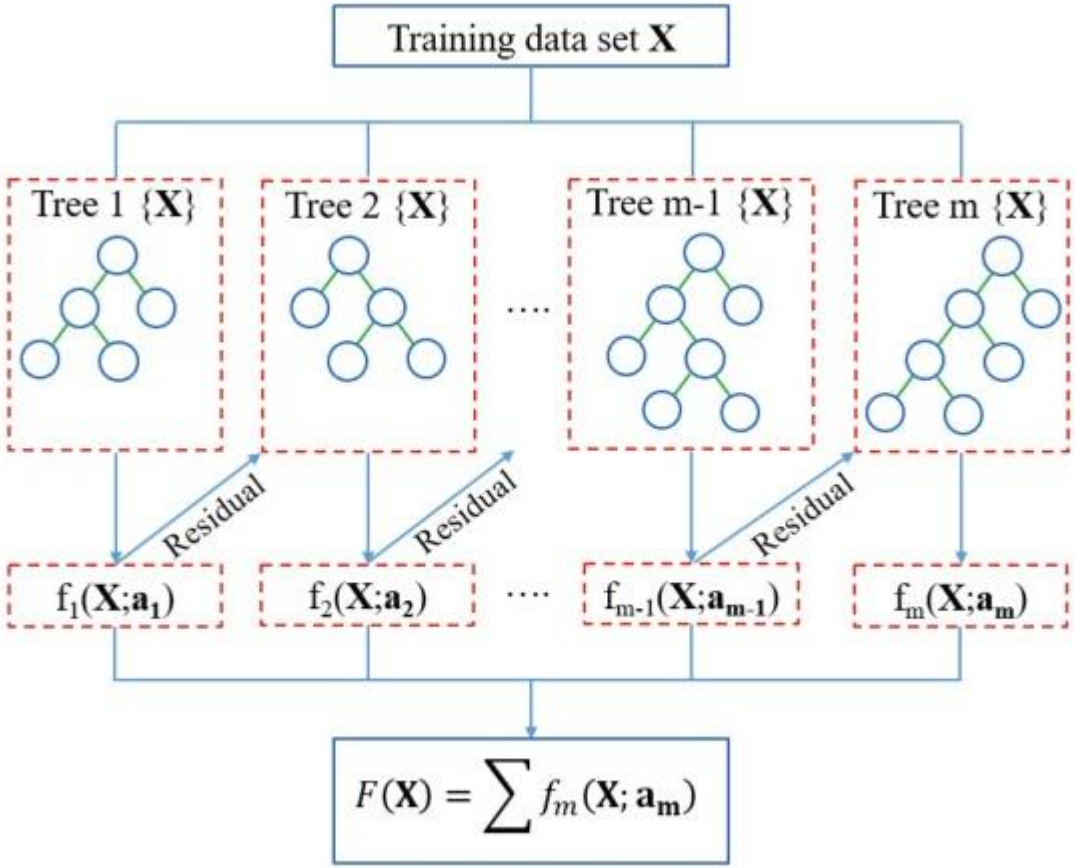
XG BOOST ALGORITHM
- eXtreme Gradient Boosting

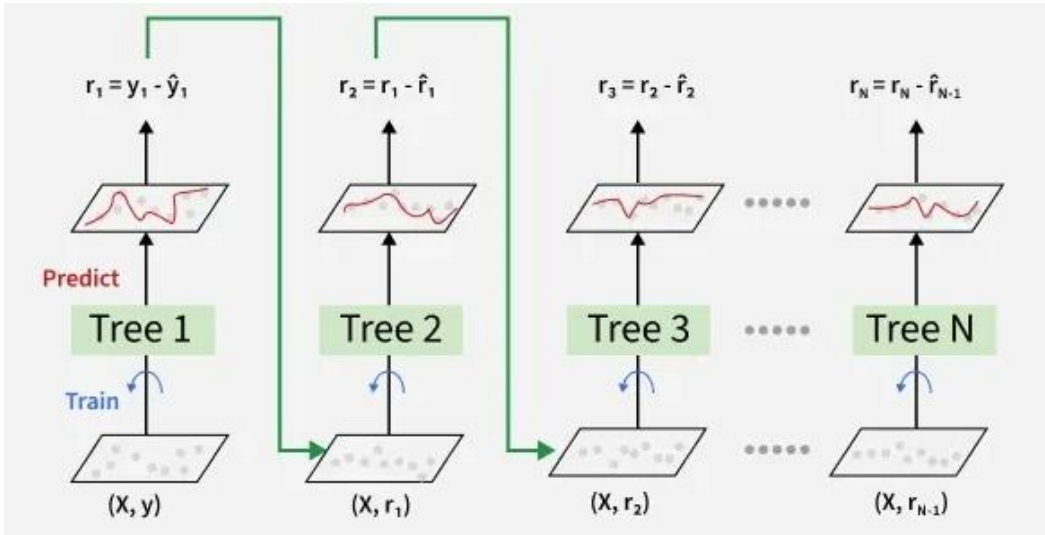
- XGBoost, which stands for eXtreme Gradient Boosting, is a **powerful and popular open-source machine learning algorithm** based on gradient-boosted decision trees. It is widely used for both classification and regression tasks due to its **efficiency, speed, and high performance**.
- **Flexibility and Hyperparameter Tuning:**

XGBoost offers a wide range of hyperparameters that can be tuned to optimize model performance for specific problems, including **learning rate, tree depth, and regularization parameters.**

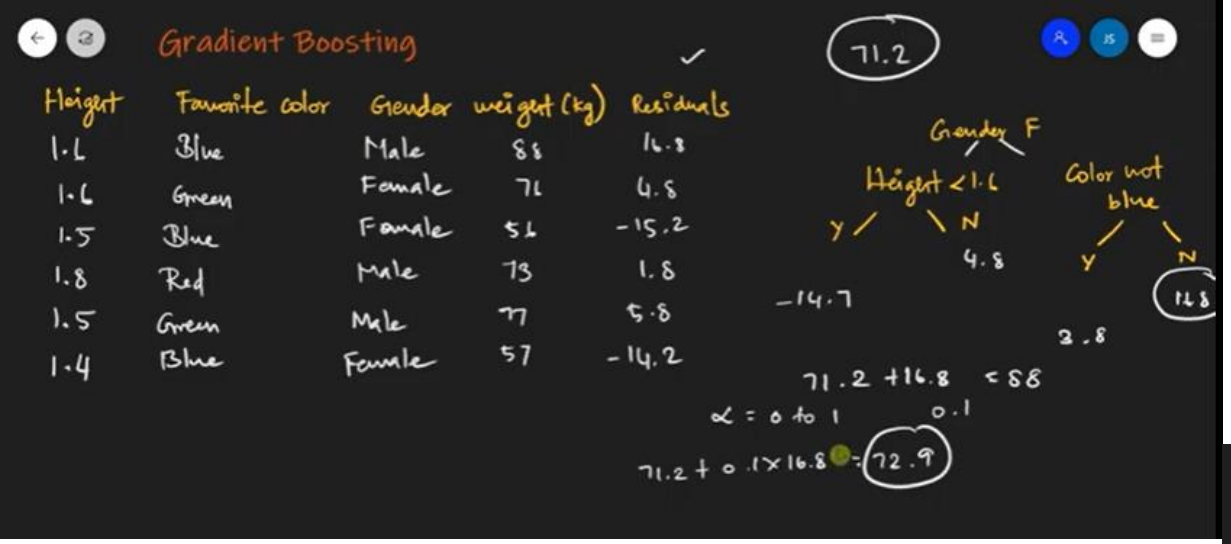
Features supported by XG-Boost.



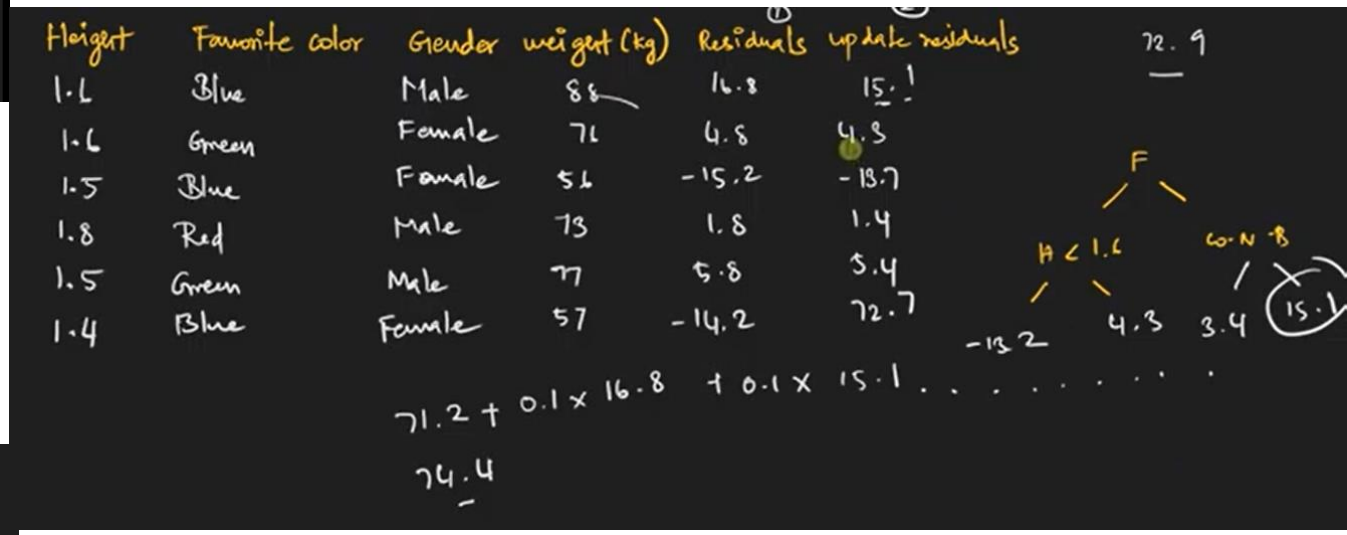




https://www.enjoyalgorithms.com/blog/xg-boost-algorithm-in-ml



Formula :
Y = old probability + (learning_rate * new prediction)



For 1st row Data:-
1st prediction = 71.2 (using mean of weight)
2nd prediction = 72.9 (using formula)
3rd prediction = 74.4 (using formula)
…..

Math behind XG Boost Algorithm

1) Find Mean
2) Find gradient
3) Find hessian
4) Find leaf weight
5) updating Probability
6) Using formula

$$\bar{y} = \frac{62 + 58 + 32 + 31 + 50 + 55 + 6}{7} = 42$$

$$g_i = \frac{\partial L}{\partial \hat{y}} = \hat{y} - y \qquad h_i = \frac{\partial^2 L}{\partial \hat{y}^2} = 1$$

$$w_j = -\frac{\sum_i g_i}{\sum_i h_i + \lambda}$$

$$\hat{y}^{(1)} = \hat{y}^{(0)} + \eta \times w$$

y	$\hat{y}_0$=42	g = ŷ - y	h = 1
62	42	-20	1
58	42	-16	1
32	42	+10	1
31	42	+11	1
50	42	-8	1
55	42	-13	1
6	42	+36	1

Where η · Learning rate

Dataset example

We want to **predict Age** based on how many sweets a person eats per week.

Person	Sweets/week	Age (target output)
A	2	14
B	4	16
C	6	20
D	8	24

Step 1: **Average of age => (14 + 16 + 20 + 24) / 4 = 18, Find the residual (using formula)**

Person	Actual Age	Initial Pred	Error (residual)
A	14	18	-4
B	16	18	-2
C	20	18	+2
D	24	18	+6

Step 2 → Build Tree 1 to model the errors, Step 3 → Compute Leaf Weights

If sweets <= 5 → younger
If sweets > 5 → older

Leaf1 (<=5 sweets): -3
Leaf2 (>5 sweets): +4

Step 4 → Update predictions

Person	Old Pred	Tree1 Update	New Pred
A (2 sweets)	18	-3	15
B (4 sweets)	18	-3	15
C (6 sweets)	18	+4	22
D (8 sweets)	18	+4	22

New residuals (remaining errors)

Person	Actual	New Pred	New Error
A	14	15	-1
B	16	15	+1
C	20	22	-2
D	24	22	+2

ROUND 2 – Iteration of tree

Person	Old Pred	Tree2	New Pred
A	15	-0.5	14.5
B	15	-0.5	14.5
C	22	+1	23
D	22	+1	23

Final Prediction

A = 14.1
B = 15.3
C = 20.9
D = 23.7

XG Boost coding in python :

My CODE :

https://github.com/krthiksha/Machine-Learning-Regression_module/blob/main/Boosting%20Algorithm%20-%20Regression/XGBoost_insurance_charge_prediction.ipynb

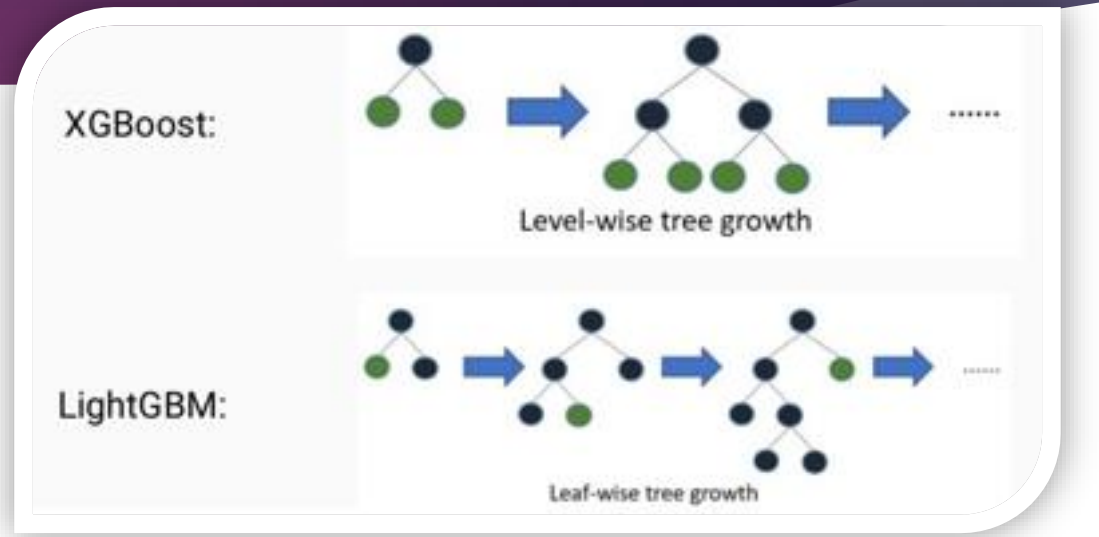
Documentation :

https://xgboost.readthedocs.io/en/latest/python/python_api.html#module-xgboost.sklearn

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html

LG boost Algorithm

- **LightGBM is a machine learning algorithm used for classification and regression.**
  It belongs to a family called **Gradient Boosting** algorithms.
- LightGBM is developed by Microsoft and is known for:
  ✓ Very fast training
  ✓ Works well on large datasets
  ✓ High model accuracy
  ✓ Memory-efficient
- **Key Features:**
  - ✓ **Histogram-Based Learning**
  - ✓ **Leaf-wise Tree Growth**
  - ✓ **Gradient-based One-Side Sampling (GOSS)**
  - ✓ **Exclusive Feature Bundling (EFB)**
  - ✓ **Support for Categorical Features**



	Accuracy:	Execution Time(in seconds):
LightGBM	0.993528	29.306975
XGBoost	0.983172	203.386410

LightGBM is 7.0 times faster than XGBOOST Algorithm

LG boost Algorithm

LightGBM Uses

- You use LightGBM when:
  - Your **dataset is large** (millions of rows).
  - Your features are numeric/categorical.
  - You want **high accuracy quickly**.
  - You need **fast training** (XGBoost often slower).

- It is widely used in:
  - Fraud detection
  - Stock price prediction
  - Recommendation systems
  - Customer churn prediction
  - Kaggle competitions

Math Behind it

Loss (regression)

$$L = \sum (y - \hat{y})^2$$

Gradient

$$\nabla = -2(y - \hat{y})$$

Tree learns gradients

Update

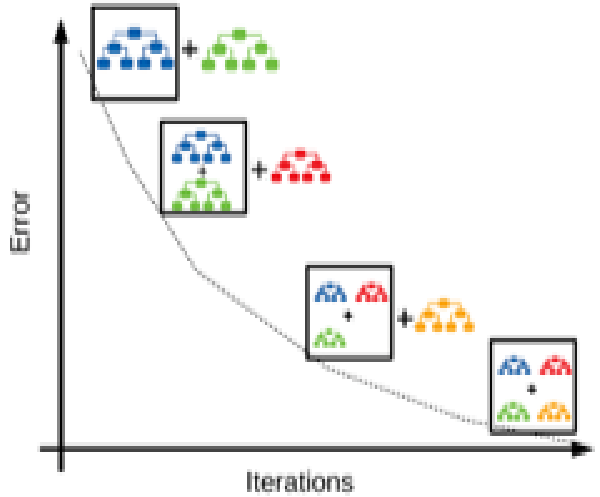
$$\hat{y}_{k+1} = \hat{y}_k + \eta \cdot h_k(x)$$

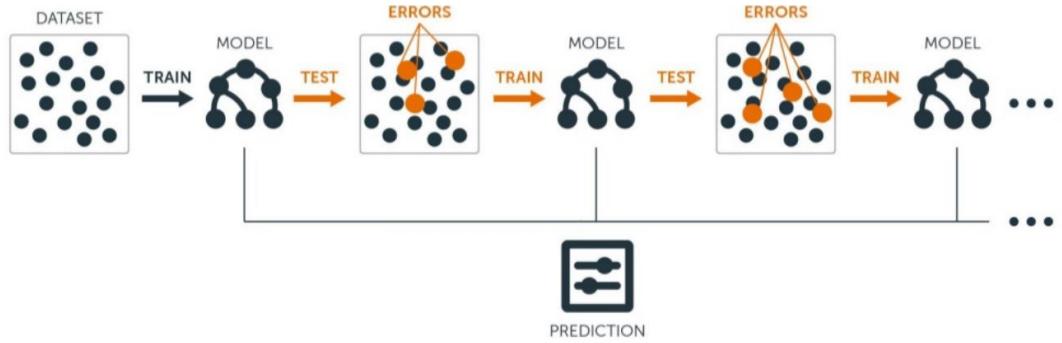
Where:

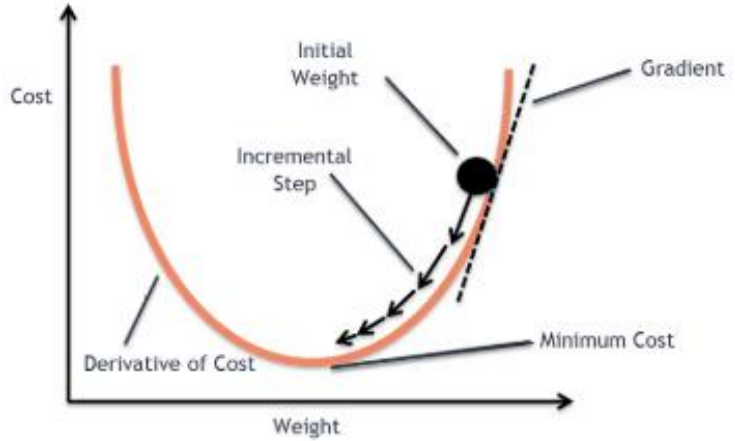
- η = learning rate
- $h_k(x)$ = tree prediction

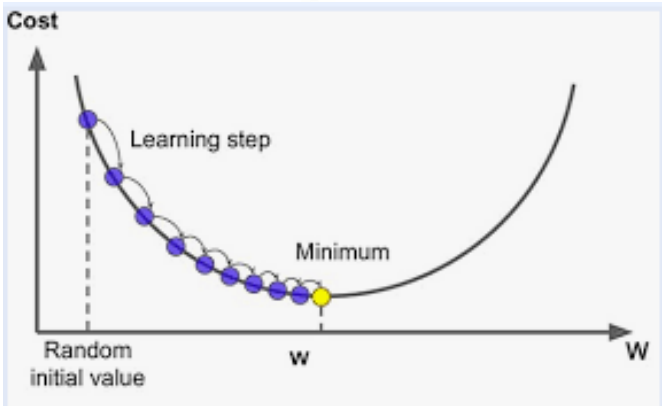
Cycle repeats.

LG Boost & Boosting concept









Dataset example

Dataset - House Price Prediction

Area	Price
1000	50
1500	70
2000	90

Step 1 — Initial Prediction

The **mean of target values** (Regression default)

$$\hat{y}_0 = \frac{50 + 70 + 90}{3} = 70$$

Sample	True	Initial Pred	Error (residuals)
S1	50	70	-20
S2	70	70	0
S3	90	70	+20

Step 2 — Compute Gradients

Gradient w.r.t prediction:

$$\nabla = -2(y - \hat{y})$$

$$y = 50, \hat{y} = 70$$

$$\nabla = -2(50 - 70) = -2(-20) = 40$$

Sample	True	Pred	Residual	Gradient
S1	50	70	-20	+40
S2	70	70	0	0
S3	90	70	+20	-40

Step 3 — Train Tree on Gradients

If Area < 1500 → +40
Else If Area between 1500 and 1800 → 0
Else → −40

Step 4 — Update Prediction

Sample	True	New Pred
S1	50	74
S2	70	70
S3	90	66

Step 5 — Repeat

New residuals → new gradients → new tree.
Each tree reduces error.
Eventually predictions get close to real values.

LightGBM updates:

$$\hat{y}_1 = \hat{y}_0 + \eta \cdot \text{tree output}$$

Where **η = learning_rate** (small step 0.1–0.3 typical)

Let η = 0.1

Now update sample predictions:

S1 (Area 1000)

$$70 + 0.1(40) = 70 + 4 = 74$$

S2 (Area 1500)

$$70 + 0.1(0) = 70$$

S3 (Area 2000)

$$70 + 0.1(-40) = 70 - 4 = 66$$

LightGBM Boost coding in python :

My CODE :

https://github.com/krthiksha/Machine-Learning-Regression_module/blob/main/Boosting%20Algorithm%20-%20Regression/LightGBM_Boost_insurance_charge_prediction.ipynb

Documentation :
https://lightgbm.readthedocs.io/en/stable/Python-API.html

https://lightgbm.readthedocs.io/en/stable/pythonapi/lightgbm.LGBMRegressor.html#lightgbm.LGBMRegressor

“Thank You”