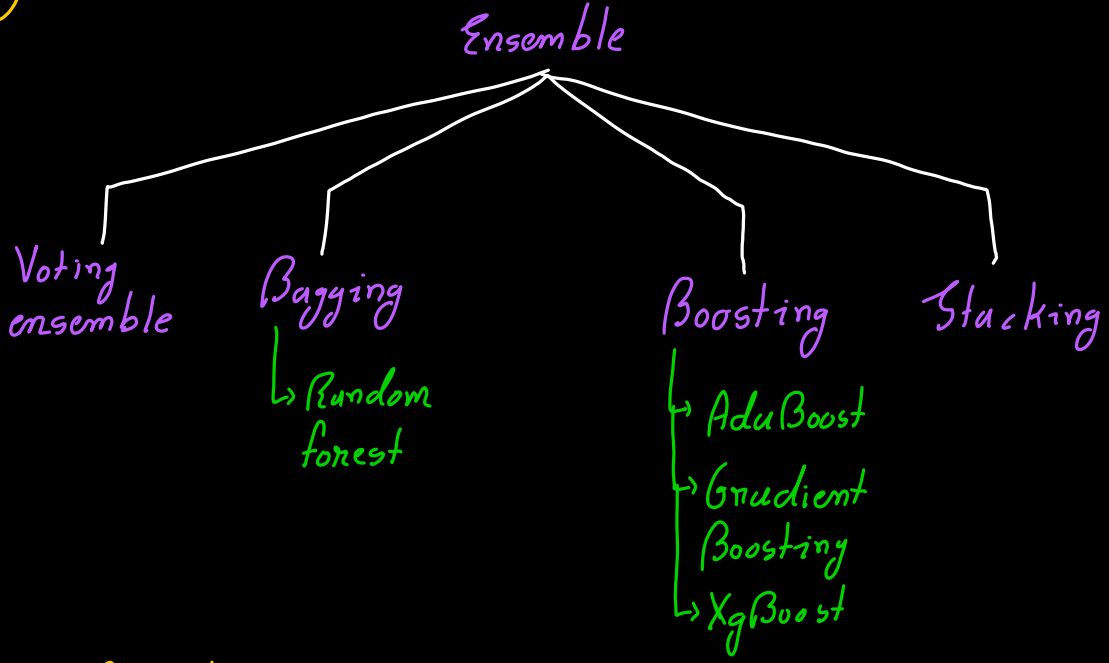
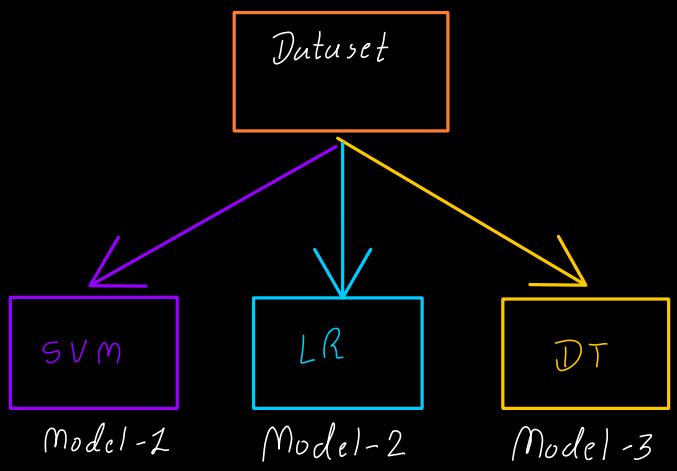
Ensemble learning

→ Many are smarter than few.



1. Voting Ensemble

→ Base models are different algorithms



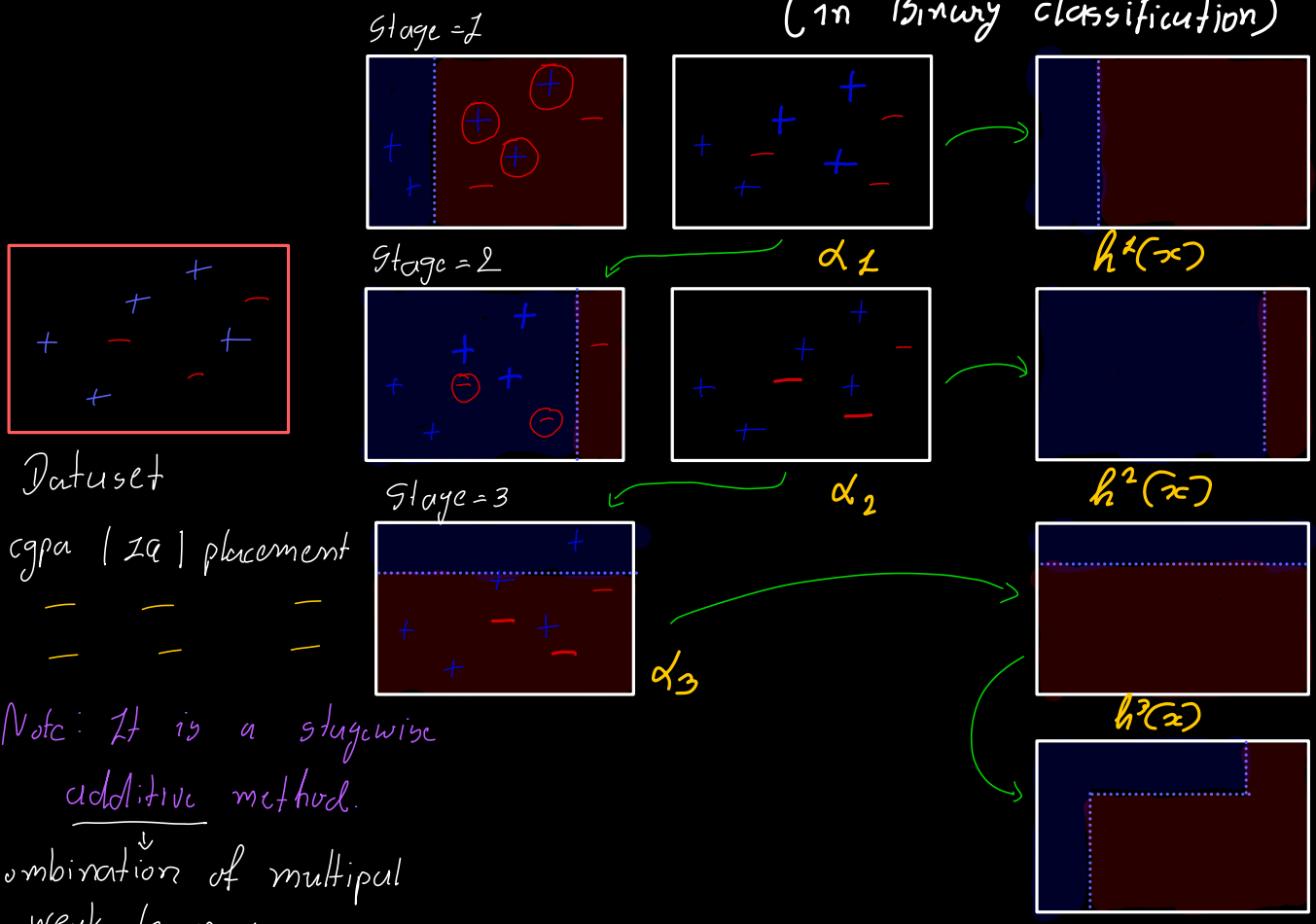
→ For classification ⟶ Majority Count

→ For regression ⟶ Average

Created with Notewise

AdaBoost Classifier

1. Weak learners
   - An ML model whose accuracy is very weak (just around 50%.)
2. Decision Stumps
   - A decision tree whose max_depth = 1
3. +1 & -1
   - Output will be +1 (+ve) or -1(-ve) instead of 0 or 1 (in Binary classification)



Note: It is a stagewise additive method.

combination of multipul weak learners.

→ There can be more number of stages.

→ α_i: A weight assigned to each model based on their performance.

→ Now combine all stages

$$H(x) = \text{Sign}(\alpha_1 * h^1(x) + \alpha_2 * h^2(x) + \alpha_3 * h^3(x))$$

Created with Notewise

→ Calculate sign of entire equation, hence +ve output will be converted to '1' & -ve will be '-1'.

→ Initially 'α' will be assigned as $1/n$ where n = num. of samples.

x_1	x_2	Y	$\hat{Y}$ (sample)	weights
3	7	1	1	0.2
2	9	0	0	0.2
1	4	1	0	0.2
9	8	0	0	0.2
3	7	0	0	0.2

$n = 5$, weights $= 1/n = 1/5 = 0.2$

Stage = 1

→ Train a Decision tree on training data, whose max_depth = 1.

creating decision stumps like....

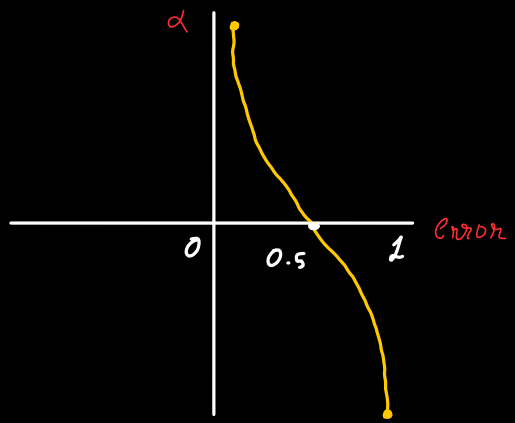
1. $x_1 > 5$ 2. $x_2 < 10$

→ Select those stumps, by which entropy decreases maximum or maximum information gain.

Suppose we have selected 1st decision stump as M_1 which gives maximum information.

→ Calculate α_1 (for M_1) which will depend on your "error rate"

- error rate ↑ α_1 ↓
- error rate ↓ α_1 ↑



$$\alpha = \frac{1}{2} \ln\left(\frac{1-\text{error}}{\text{error}}\right)$$

Error = Sum of weights of all mis-classified rows.

$$\alpha_1 = \frac{1}{2} \ln\left(\frac{1-\text{error}}{\text{error}}\right)$$

$$\frac{1}{2} \ln\left(\frac{1-0.2}{0.2}\right)$$

$$\frac{1}{2} \ln\left(\frac{0.8}{0.2}\right)$$

$$\alpha_1 = \boxed{0.7}$$

Up-sampling

→ Increasing the weight of wrongly classified rows

→ Decreasing the weight of correctly classified rows

new_weight = current_weight * e^{α_1}

new_weight = current_weight * $e^{-\alpha_1}$

Created with Notewise

$W_{correct} = 0.2 * e^{-0.7} = \boxed{0.0994}$

$W_{wrong} = 0.2 * e^{0.7} = \boxed{0.4026}$

Gradient Boosting

→ Process of making a powerfull ML model by combining multipul weak models.

→ Process goes in sequence, where first model will be trained on data

→ The task of next model is to correct the mistakes made by previous model.

→ If we are working with regression problem in gradient boosting then the o/p of first model will always be the mean of output column.

→ Actually the first node is not a ML model. it is just a mean calculator which always return mean of output variable, No matter what-ever input is given.

(in G.B. terms)

$Cost = y - \hat{y} = \text{pseudo residual}$

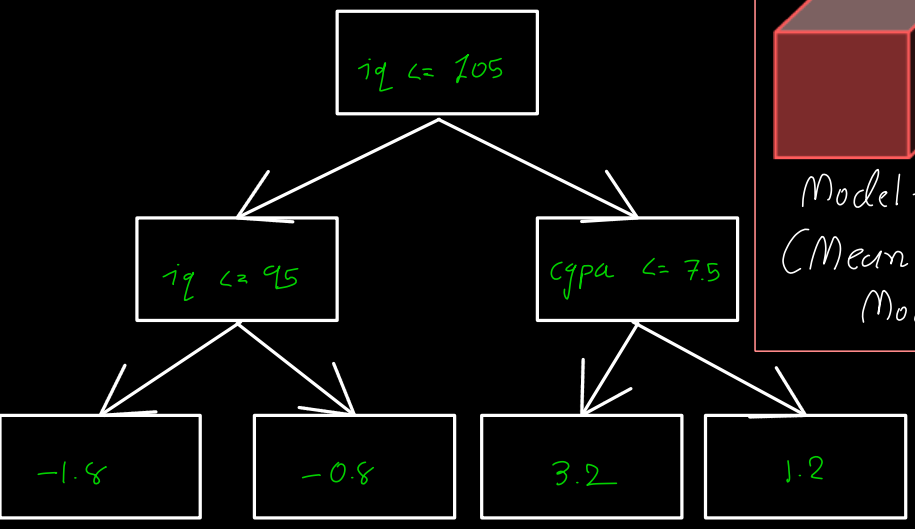
IQ	CGPA	salary	Prediction 1 (m1)	Pseudo residual m1	Prediction 2 (m2)
90	5	3	4.8	-1.8	-1.8
100	7	4	4.8	-0.8	-0.8
110	6	8	4.8	3.2	3.2
120	9	6	4.8	1.2	1.2
80	5	3	4.8	-1.8	-1.8

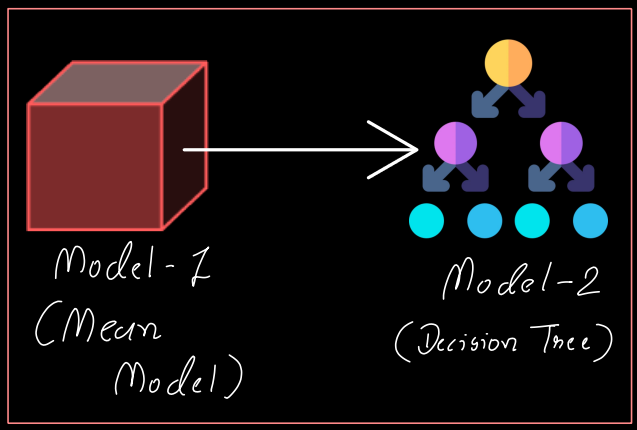
These mistakes of model-1 will be given to model-2 as a 'target_variable'. Now model-2 will predict these mistakes based on given i/p 'cgpa' & 'iq'.

→ In G.B. generally the second model is always Decision_Tree due to its performance in G.B.

Created with Notewise

→ This is how model-2 looks like.





→ Generally we includes α in prediction to reduce overfitting.

$\alpha = 0.1$

Prediction = Prediction-1 + α * Prediction-2

Residual-2 (M2)	actual − predicted
-1.62	3 − (4.8 + 0.1 * (-1.8)) = -1.62
-0.72	4 − (4.8 + 0.1 * (-0.8)) = -0.72
2.48	8 − (4.4 + 0.1 * (3.2)) = 2.68
1.08	6 − (4.8 + 0.1 * (7.8)) = 1.08
-1.62	3 − (4.8 + 0.1 * (-7.8)) = -1.62

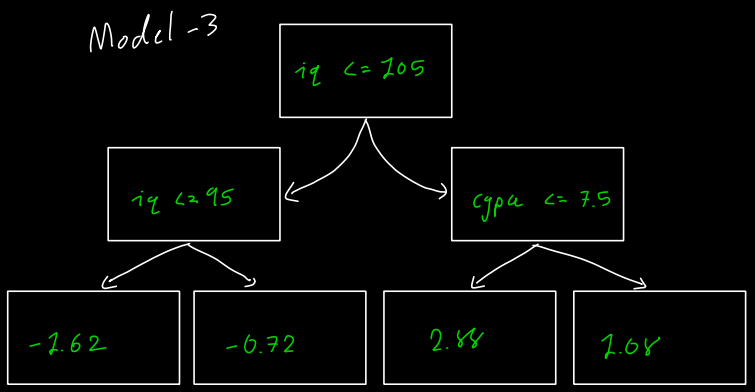
→ As we add models residual will move towards 0, 4 will keep mooving.

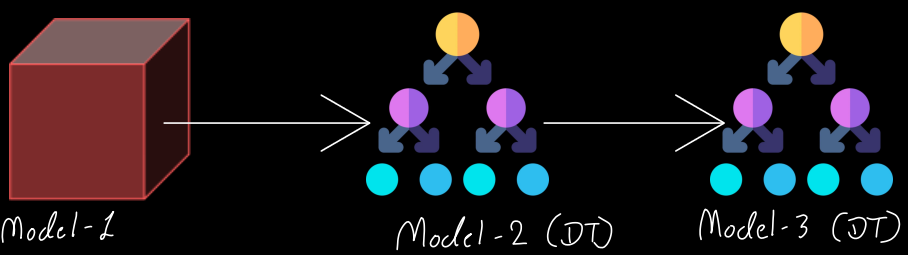
→ For Model_3 input will be cgpa, iq but this model will predict the mistakes of first & second models.

→ This process will continue for all n number of models, linked in this chain.

Created with Notewise

Prediction-3 (M3)	Residual-3 (M3)
-1.62	1.45
-0.72	-0.65
2.88	2.6
1.08	0.97
-1.62	1.46





Residual_3 calculation

actual - predicted

3 - (4.8 + 0.1 * (-1.8) + 0.1 * (-1.62)) = 1.45

4 - (4.8 + 0.1 * (-0.8) + 0.1 * (-0.72)) = -0.65

4 - (4.8 + 0.1 * (3.2) + 0.1 * (2.88)) = 2.6

6 - (4.8 + 0.1 * (1.2) + 0.1 * (1.08)) = 0.97

3 - (4.8 + 0.1 * (-1.8) + 0.1 * (-1.62)) = -1.46

AdaBoost vs GradientBoost

1 : max_leaf_node

[a]. AdaBoost
- ↳ We uses desicion stumps
- ↳ Each tree will have maximum depth 1 & maximum leaf nodes = 2

[b]. GradientBoosting
- ↳ This is recommended that in regression task, max_leaf_node should be 8 to 32.

2. Learning_rate

[a]. AdaBoost
- ↳ Weight will be assigned to each model, based on their performance. (High performance = more weight)

Created with Notewise

[1]. GradientBoost

↳ Single constant weight will be multiplied with all model's prediction value.

Gradient Boosting Classifier

-> Majority of process will be similar to GB regressor

-> Process will slightly different to calculate initial prediction and class assignment.

---

	cgpa	iq	is_placed
0	6.82	118	0
1	6.36	125	1
2	5.39	99	1
3	5.50	106	1
4	6.39	148	0
5	9.13	148	1
6	7.17	147	1
7	7.72	72	0

-> for 1st model, the probability will be calculated using log(odd) method.

$$\log(\text{odd}) = \log\left(\frac{\text{number of 1s in o/p column(Y)}}{\text{number of 0s in o/p column(Y)}}\right)$$

$$= \log\left(\frac{5}{3}\right)$$

$$= \boxed{0.51}$$

	cgpa	iq	is_placed	pre1(log-odds)
0	6.82	118	0	0.510826
1	6.36	125	1	0.510826
2	5.39	99	1	0.510826
3	5.50	106	1	0.510826
4	6.39	148	0	0.510826
5	9.13	148	1	0.510826
6	7.17	147	1	0.510826
7	7.72	72	0	0.510826

-> Now we need to calculate pseudo_residual (actual - predicted)

-> But we can't directly compute residual by subtracting pre1(log-odds) from is_placed because is_placed shows probability & pre1(log-odds) has log values.

-> Hence, we will convert pre1(log-odds) into probability using sigmoid.

$$\text{probability} = \left(\frac{1}{1 + e^{-\log(\text{odds})}}\right)$$

	cgpa	iq	is_placed	pre1(log-odds)	pre1(probability)
0	6.82	118	0	0.510826	0.625
1	6.36	125	1	0.510826	0.625
2	5.39	99	1	0.510826	0.625
3	5.50	106	1	0.510826	0.625
4	6.39	148	0	0.510826	0.625
5	9.13	148	1	0.510826	0.625
6	7.17	147	1	0.510826	0.625
7	7.72	72	0	0.510826	0.625

-> Now probability values are > 0.5 (assumed threshold) residual_1 = (is_placed - pre1(probability))

-> In short, we have replaced mean calculation process (GBR for first model only) with log(odds) for GB classification tasks.

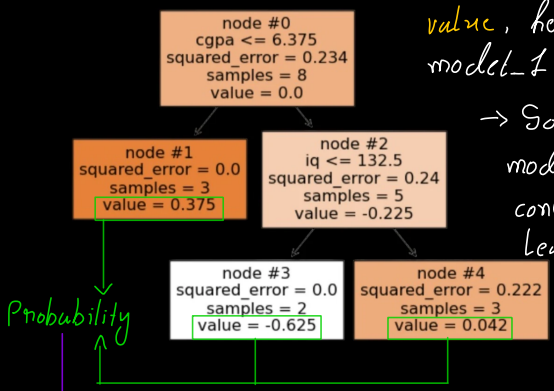
Created with Notewise

-> Now we will train a new DecisionTreeRegressor model in which input will be cgpa, iq and target variable will be residual_1.

-> This is how model_2 looks like.

-> The leaf node of these decision tree returns probability value, hence we can't directly compare it with o/p of model_1 (log-odd value)

-> So pre1(log-odds) are in log formate & leaf node of model_2 are in probability formate so we need to convert these probability in log formate. for all leaf node output values using below formula.

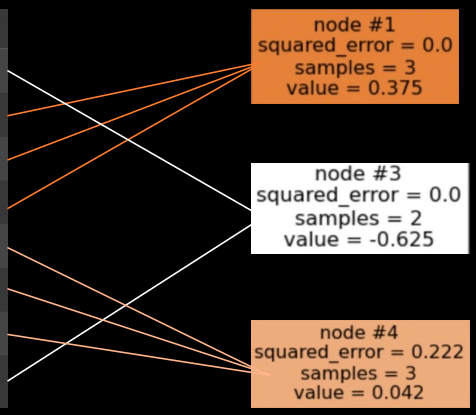


calculate log-odds value for each probability values.

$$\text{log odds} = \frac{\sum \text{Residual}}{\sum [\text{previous\_prob} * (1 - \text{Previous\_prob})]}$$

-> Now we will extract the Bucket address of each row.
(which row is passed through which decision leaf node)

	cgpa	iq	is_placed	pre1(log-odds)	pre1(probability)	res1	leaf_entry1
0	6.82	118	0	0.510826	0.625	-0.625	3
1	6.36	125	1	0.510826	0.625	0.375	1
2	5.39	99	1	0.510826	0.625	0.375	1
3	5.50	106	1	0.510826	0.625	0.375	1
4	6.39	148	0	0.510826	0.625	-0.625	4
5	9.13	148	1	0.510826	0.625	0.375	4
6	7.17	147	1	0.510826	0.625	0.375	4
7	7.72	72	0	0.510826	0.625	-0.625	3



log odds for #node 3 (-0.625, -0.625)

$$= \frac{-0.625 - 0.625}{0.625(1-0.625) + 0.625(1-0.625)}$$

$$= \frac{-2 \times 0.625}{0.685 \times 0.375 + 0.625 \times 0.375}$$

$$= \frac{-2 \times 0.625}{2 \times 0.625 \times 0.375}$$

$$= \frac{-1}{0.375}$$

$$= -2.66$$

node #1 = 1.6

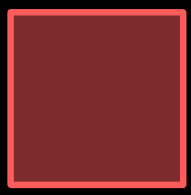
node #3 = -2.66

node #4 = 0.18

Created with NoteWise

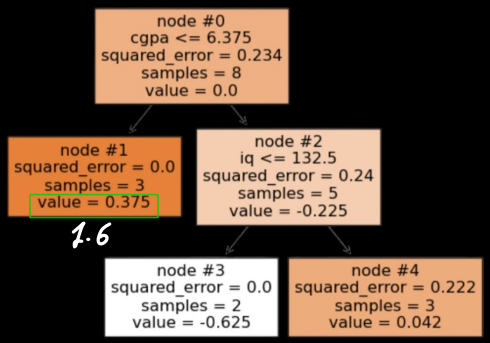
→ Now we will combine the o/p of both models.

→ Overall o/p log-odds for each row.



0.51

+



1.6 -2.66 0.18

	cgpa	iq	is_placed	pre1(log-odds)	pre1(probability)	res1	leaf_entry1
0	6.82	118	0	0.510826	0.625	-0.625	3
1	6.36	125	1	0.510826	0.625	0.375	1
2	5.39	99	1	0.510826	0.625	0.375	1
3	5.50	106	1	0.510826	0.625	0.375	1
4	6.39	148	0	0.510826	0.625	-0.625	4
5	9.13	148	1	0.510826	0.625	0.375	4
6	7.17	147	1	0.510826	0.625	0.375	4
7	7.72	72	0	0.510826	0.625	-0.625	3

$0.51 + (-2.66) =$
$0.51 + (1.6) =$
$0.51 + (1.6) =$
$0.51 + (1.6) =$
$0.51 + (0.18) =$
$0.51 + (0.18) =$
$0.51 + (0.18) =$
$0.51 + (-2.66) =$

pre2(log-odds)
-2.159174
2.110826
2.110826
2.110826
0.690826
0.690826
0.690826
-2.159174

convert back to probability

$$\frac{1}{1+e^{-\text{log odd}}}$$

	cgpa	iq	is_placed	pre1(log-odds)	pre1(probability)	res1	leaf_entry1	pre2(log-odds)	pre2(probability)	res2	leaf_entry2
0	6.82	118	0	0.510826	0.625	-0.625	3	-2.159174	0.103477	-0.103477	3
1	6.36	125	1	0.510826	0.625	0.375	1	2.110826	0.891951	0.108049	3
2	5.39	99	1	0.510826	0.625	0.375	1	2.110826	0.891951	0.108049	3
3	5.50	106	1	0.510826	0.625	0.375	1	2.110826	0.891951	0.108049	3
4	6.39	148	0	0.510826	0.625	-0.625	4	0.690826	0.666151	-0.666151	4
5	9.13	148	1	0.510826	0.625	0.375	4	0.690826	0.666151	0.333849	2
6	7.17	147	1	0.510826	0.625	0.375	4	0.690826	0.666151	0.333849	2
7	7.72	72	0	0.510826	0.625	-0.625	3	-2.159174	0.103477	-0.103477	2

$y - \hat{y} =$

→ Here we have noticed that the error is decreasing sharply it clearly indicates faster learning process.

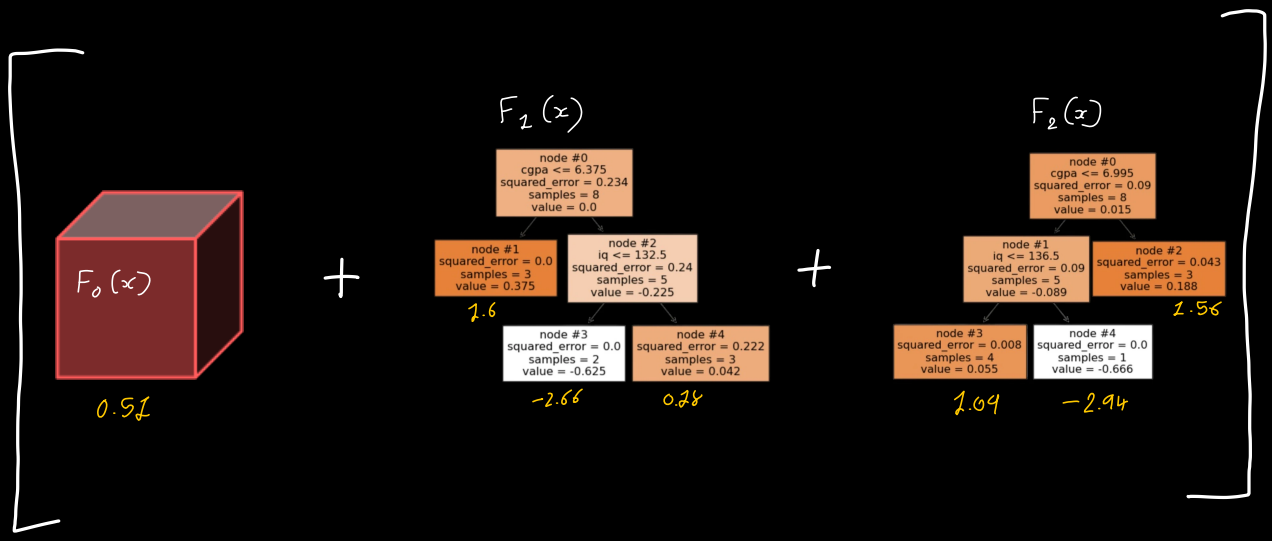
→ But we should not go with such amount of acceleration during learning.

→ In order to reduce learning speed, we will use α (learning rate)

Created with Notewise

by multiplying with each log-odd values.

Now let's add one more model



	cgpa	iq	is_placed	pre1(log-odds)	pre1(probability)	res1	leaf_entry1	pre2(log-odds)	pre2(probability)	res2	leaf_entry2
0	6.82	118	0	0.510826	0.625	-0.625	3	-2.159174	0.103477	-0.103477	3
1	6.36	125	1	0.510826	0.625	0.375	1	2.110826	0.891951	0.108049	3
2	5.39	99	1	0.510826	0.625	0.375	1	2.110826	0.891951	0.108049	3
3	5.50	106	1	0.510826	0.625	0.375	1	2.110826	0.891951	0.108049	3
4	6.39	148	0	0.510826	0.625	-0.625	4	0.690826	0.666151	-0.666151	4
5	9.13	148	1	0.510826	0.625	0.375	4	0.690826	0.666151	0.333849	2
6	7.17	147	1	0.510826	0.625	0.375	4	0.690826	0.666151	0.333849	2
7	7.72	72	0	0.510826	0.625	-0.625	3	-2.159174	0.103477	-0.103477	2

pre3(log-odds)	pre3(probability)
-1.068349	0.255717
3.201651	0.960896
3.201651	0.960896
3.201651	0.960896
-1.798349	0.142052
2.251651	0.904793
2.251651	0.904793
-0.598349	0.354722

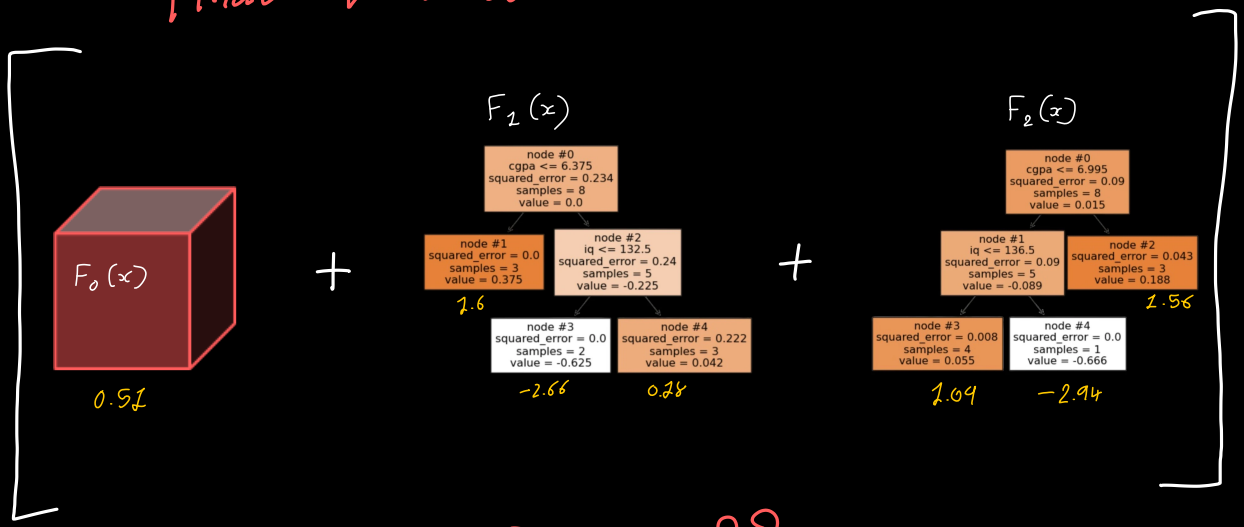
Now the training process is over, let's see how this model does prediction over unseen data.

* Steps

1. calculate log-odd values.
2. Calculate probability of those log-odd values.
3. O/p based on probability.

Created with Notewise

Final trained model.



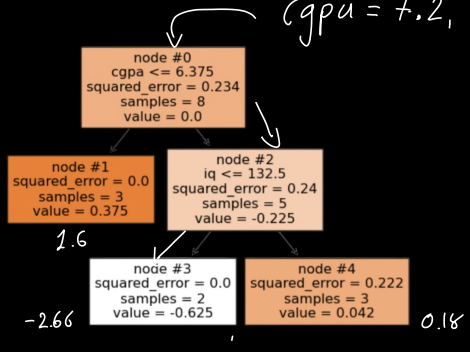
query = [[7.2 , 100]] → {0 , 1} ??

[a]. Model_1

log-odds = 0.51

[b]. Model_2

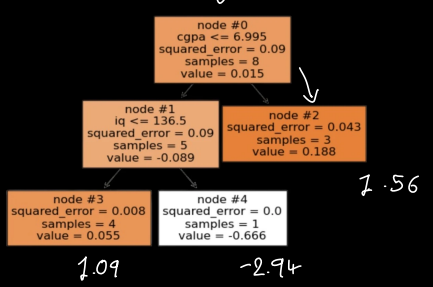
cgpa = 7.2, iq = 100



log-odds = -2.66

[c]. Model_3

cgpa = 7.2 iq = 100



log-odds = 1.56

= 0.51 + (-2.66) + (1.56)

= **-0.59** log-odds

probability (0.59) = **0.35**

assumed threshold = 0.5

o/p = 0

Created with Notewise

XGBoost (Extreme Gradient Boosting)

→ A ML model which can play very excellent on any kinds of dataset. No matter what is the length of characteristics of dataset.

→ XGBoost is not any ML algorith but it is a library - made over Gradient Boost with including some principals of software development.

Q: Why GradientBoost...?

1. flexible
   - ↳ we can use any loss-functions.
   - ↳ we can use it for any kinds of problem like Regression, classification, Ranking, custom problems.
2. Performance
3. Robust
4. Most famous on kaggle competition.

- Core uneus of XGBoost
  - Performance
  - Speed
  - Flexibility

→ Cross platform support
→ Multi-language support
- Java
- Scala
- Ruby
- Python
- R
- Swift
- Julia
- C/C++

→ Integration with more libraries & tools.
→ Supports all kinds of ML problems.

1. Performance

Compatibility

- Python libraries
  - Numpy
  - Pandas
  - Matplotlib
  - Scikit-learn
- Distributed Computing
  - Spark
  - Pyspark
  - Dask
- Model interpretability
  - SHAP
  - Lime
- Model deployment
  - Docker
  - kubernetes
- Workflow management
  - Airflow
  - MLflow

Usecases

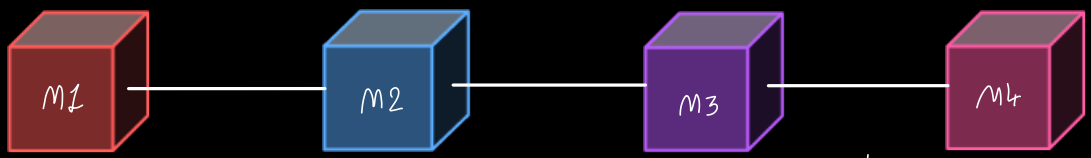
→ Regression, Classification (Binary + multi-class), Time series forecasting, Ranking Problems, Anomaly Detection

Created with Notewise

2. Speed

[a.] Parallel processing

- As we know in Gradient boosting, model building is sequential process.



- The process will be is sequence, one model at a time.
- As we know the process of building decision tree, where we try to find best splitting criteria.

Suppose we have a dataset with 2 i/p & 1 o/p variables like

CGPA	IQ	placement
7.5	99	1
9.0	30	0
6.7	100	1

- Now in order to get root node, we calculates best split of each colum. after this process, we selects best one split from all these best splits of each columns.
- XGboost uses parallel calculation mechanism in which it calculates best split of each colum symontenicously & then selects best one. hence the time consumption is less.

[b]. Optimized Data Structure

- Generally ML models stores data row wise and processes data row by row.
- xgboost uses column block data structure in which data is stored in column format.
- Each block will be a single column.

[c] cache awareness

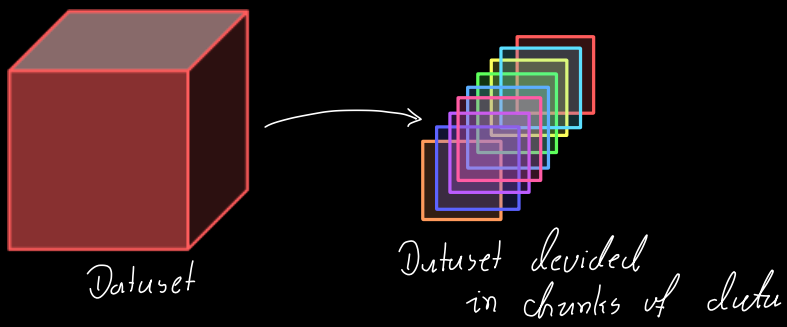
- Efficiently uses cache memory.
- XGBoost uses histogram based training.
- XGBoost will create histogram for each column block. and it stores range value of each bins in cache memory. Because those bin's range value will be used frequently by model.

[d]. Out of core computing

Created with Notewise

↳ Data will be divided into chunks and then sequentially it loads each chunk of data.

Lets suppose u have a laptop with 8 GB RAM, and you want to work with a dataset whose size is 15 GB.
in this situation, XGBoost is here to help. by deviding dataset to small chunks.



[e]. Distributed Computing

↳ The process of deviding a single task between multipal machines.
↳ For XGBoost, we will distribute data between multipal machines & train the models parallelly and in last we will aggregate all models.
for this, we need to use "DASK" or "kubernetes"

[f]. GPU support

3. Performance

[a]. Regularizing learning objective

↳ In GB we does regularization using learning rate and pruning, but in XGBoost, it has in-build regularization feature in its loss function

[b]. Sparsity Aware Split finding

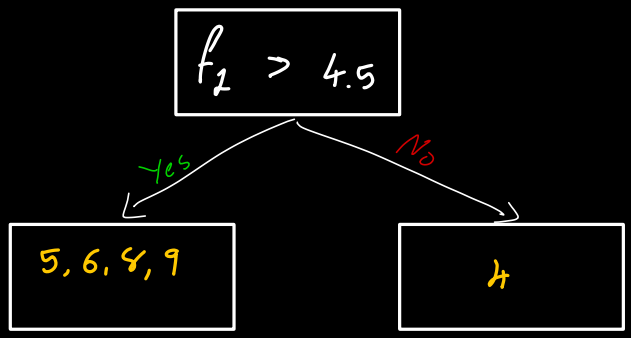
↳ Sparsity : A dataset which contains too many zeros.
↳ XGBoost has capability to identify and handle sparsity.
Lets see how XGBoost does.

Created with Notewise

Sample dataset

f_1	gini entropy
4	
	— 4.5
5	
	— 5.5
6	
NuN	— 7
8	
	— 8.5
9	

best split
$f_1 > 4.5$



→ Now based on above split, algorith will put 'NuN' valued row in both leaf node one-by-one and calculate 'information-gain' which-ever node has higher 'information-gain', 'NuN' will be placed in that particular leaf node.

[c] Handling Missing Values

↳ Handles missing values internally

[d]. Efficient split finding (Weighted Quantile Sketch + Approximate Tree Learning)

f_1
3
5 — 4
7 — 6
9 — 8
11 — 10
13 — 12

→ we calculates 'gini-entropy' using "exact-greedy-search" method.
→ But as we can see, this approach is very slow
→ To solve this problem Approximate Tree learning approach comes into picture.

Approximate Tree Learning

↳ Convert this numerical column dataset into bins (like [1-5], [6-10], [11-15] .... (histogram based training)
↳ Now this continues variable became descreate variable.
↳ This bins creation process is called as "histogram-based-training" for this process we uses a technique is called as Weighted Quantile Sketch
↳ We selects best bin range based on the distribu-

Created with Notewise

tion of the variable.

↳ We should use "Quantile" approach to create bins ensuring each bin contains equal number of data points

[e]. Tree Pruning

↳ The process of reducing the depth of a tree is called as pruning.

↳ This method can helps to reduce the complexity of model. in short 'Tree pruning' reduces overfitting.

Types of pruning

↳ Post-pruning

↳ Pre-pruning

→ There are few more algorithms like LightGBM, CatBoost for better accuracy and robust performance.

Created with Notewise