Gradient Boost

Gradient Boost algorithm belongs to the class of boosting ensemble algorithms where base models/ weak learners are trained in a sequential manner such that each new learner tries to correct the errors made by the previous ones. Gradient Boost also follows the Stagewise Additive Method for It can be used by both classification and regression.

In Gradient Boost, we use decision trees as the weak learner but here the depth of tree is not limited to 1 ie. they are not decision stumps as they were in case of AdaBoost algorithm. Here, we can decide the number of leaves our decision tree can have. which is often between 8 to 32 but can be tuned using hyper parameter tuning.
The **loss function** used in gradient boosting algorithm is MSE in case of regression problems and log-loss (or cross-entropy) in case of classification problems.

The final prediction is made by summing up the contributions of all the individual models (usually decision trees) that have been trained in sequence. Here's the general formula for making a prediction:

$$\hat{y} = F_M(x) = F_0(x) + \sum_{i=1}^{M} \gamma_i \cdot f_m(x)$$

where, → $\hat{y}$ is the final prediction for input x.
→ $F_M(x)$ is the final model after M iterations (i.e., after training M trees).
→ $F_0(x)$ is the initial prediction, which is often the mean of target values in case of regression or log-odds in case of classification.
→ $f_m(x)$ is the prediction from the m-th decision tree trained at the m^{th} iteration.

→ ν is the learning rate, a scaling factor (typically between 0 and 1) that controls the contribution of each tree to the final prediction.

→ M is the total number of trees in the ensemble.

The formula allows gradient boosting to iteratively improve the model by reducing the pseudo-residuals at each step, leading to a more accurate final prediction.

Regression using Gradient Boost

Steps for doing regression using gradient boost :-

1. Initialize $f_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, \gamma)$

2. For $m = 1$ to M : (M → total no. of trees used in the model)

   (a) For $i = 1, 2, \ldots, N$ compute

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f = f_{m-1}}$$

   (b) Fit a regression tree to the targets r_{im} giving terminal regions R_{jm}; $j = 1, 2, \ldots, J_m$

   (c) For $j = 1, 2, \ldots, J_m$ compute

$$\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$

   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$

(3) Output $\hat{f}(x) = f_M(x)$

Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$ (features → x_i, output → y_i, n → no. of rows), a differentiable loss function $L(y, F(x))$ (y → actual, $F(x)$ → predicted), number of iterations M.

Let's understand this using an example taking a small dataset:

iq	cgpa	salary
90	8	3
100	7	4
110	6	8
120	9	6
80	5	3

For the sake of simplicity and since our dataset is a toy dataset, we will be using only three weak learners.

$f_0(x)$ $f_1(x)$ $f_2(x)$

$m_1 \rightarrow m_2 \rightarrow m_3 \rightarrow$

Input: As we discussed to use gradient boost we require a training set $\{(x_i, y_i)\}_{i=1}^{n}$. In our context x_i is the features where x_1 denotes first row inputs, x_2 denotes second row in the dataset and so on. We have $n=5$ in our dataset.

$x_1 = (90, 8)$, $x_2 = (100, 7)$ and so on.

y_i denotes the output/target where y_1 denotes output of first row, y_2 denotes output of second row - - so on.

Here, $y_1 = 3$, $y_2 = 4$, $y_3 = 8$ - - - $y_5 = 3$.

Loss function ⇒ We also require a differentiable loss function which needs to be differentiable at every point.

The loss function, we use typically in regression problems is **least squares** i.e.,

$$L = \frac{1}{2} \sum_{i=1} (y_i - \hat{y}_i)^2$$

Since, we have decided to train 3 weak learners our $M = 3$.

Step 1. We have to initialize

$$f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma), \text{ replacing } L$$

$$f_0(x) = \arg\min_{\gamma} \frac{1}{2} \sum_{i=1}^{n} (y_i - \gamma)^2$$

This equation means, we want to find a value of γ such that value of loss becomes minimum. We can do it by differentiation w.r.t. γ and equating it to 0.

$$\frac{d}{d\gamma} f_0(x) = \frac{\partial}{\partial \gamma} \frac{1}{2} \sum_{i=1}^{n} (y_i - \gamma)^2$$

$$= \frac{1}{2} \sum_{i=1}^{n} \frac{\partial}{\partial \gamma} (y_i - \gamma)^2$$

$$= \sum_{i=1}^{n} (y_i - \gamma) \frac{\partial}{\partial \gamma} (y_i - \gamma) \quad \{\text{Using chain rule}\}$$

$$\frac{\partial}{\partial \gamma} f_0(x) = -\sum_{i=1}^{n} (y_i - \gamma)$$, now we need to equate this to 0 to find the minimum value.

$$\Rightarrow -\sum_{i=1}^{n} (y_i - \gamma) = 0$$

$$\Rightarrow \sum_{i=1}^{n} (\gamma - y_i) = 0$$, since in our example dataset we have $n=5$, so we can write it as-

$$\Rightarrow \sum_{i=1}^{5} (\gamma - y_i) = 0$$

$$\Rightarrow (\gamma - 3) + (\gamma - 4) + (\gamma - 8) + (\gamma - 6) + (\gamma - 3) = 0$$

$$\Rightarrow 5\gamma = 24 \Rightarrow \gamma = \frac{24}{5} = 4.8$$

iq	cgpa	salary	$f_0(x)$
90	8	3	4.8
100	7	4	4.8
110	6	8	4.8
120	9	6	4.8
80	5	3	4.8

This implies $f_0(x) = 4.8$, which is nothing but the mean of the output/ target column.

* In gradient boost, the first weak learner gives the prediction as mean value of target column and it is known as leaf. It does not have any ML model involved.

Step 2: For $m = 1$ to 3 (as M is 3 in our case)

(a) For $i = 1$ to n, compute

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f = f_{m-1}}$$

→ this means for every model (m), for every row (i), we need to calculate a **pseudo-residual**. Since we are going to use first model, so $m = 1$, hence

$$r_{i1} = -\left[\frac{\partial L(y_i, \gamma_i)}{\partial \gamma_i}\right]_{f = f_{1-1} = f_0}$$ [In our case, $\gamma = f(x)$]

Also, we have 5 rows in our dataset, so we want to calculate $r_{11}, r_{21}, r_{31}, \ldots r_{51}$.

r_{11} means for first model first row pseudo-residual.

r_{51} means for first model fifth row pseudo-residual.

Replacing the value of loss fn in the equation.

$$r_{i1} = -\left[\frac{\partial}{\partial \gamma_i} \frac{1}{2}(y_i - \gamma_i)^2\right]_{f=f_0}$$

$$\Rightarrow r_{i1} = -\left[\frac{1}{2} \times 2 (y_i - \gamma_i) \frac{\partial}{\partial \gamma_i}(y_i - \gamma_i)\right]_{f=f_0}$$

$$\Rightarrow r_{i1} = -\left[(y_i - \gamma_i)(-1)\right]_{f=f_0}$$

$$\Rightarrow r_{i1} = \left[y_i - \gamma_i\right]_{f=f_0}$$

$$\Rightarrow r_{i1} = \left[y_i - f(x_i)\right]_{f=f_0} \quad (\because \gamma = f(x))$$

$$\Rightarrow r_{i1} = \left[y_i - f_0(x_i)\right]$$

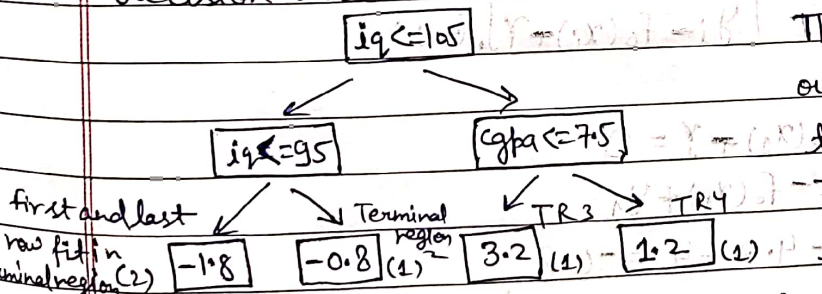
$$\therefore r_{11} = y_1 - f_0(x_1),\ r_{21} = y_2 - f_0(x_2), \ldots, r_{51} = y_5 - f_0(x_5)$$

$$\Rightarrow r_{11} = 3 - 4.8 = -1.8,\ r_{21} = 4 - 4.8 = -0.8,\ r_{31} = 8 - 4.8 = 3.2$$

$r_{41} = 6 - 4.8 = 1.2$, $r_{51} = -1.8$, therefore our dataset now is –

iq	cgpa	Salary	$f_0(x)$	r_{i1}
90	8	3	4.8	-1.8
100	7	4	4.8	-0.8
110	6	8	4.8	3.2
120	9	6	4.8	1.2
80	5	3	4.8	-1.8

(b) Now, we have to fit a regression tree using columns iq, cgpa, salary as input and r_{i1} as the target. The decision tree will look like this



This decision tree divides our data into four terminal regions. The end values we get in leaf nodes is by taking mean of all points lying in their corresponding region.

(c) Now, for all terminal regions, $j = 1$ to 4, compute

$$\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i + \gamma))$$

$$\Rightarrow \gamma_{j1} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{j1}} L(y_i, f_{1-1}(x_i + \gamma))$$

γ_{11} γ_{21} γ_{32} γ_{41} ⇒ $\gamma_{j1} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{j1}} L(y_i, f_0(x_i + \gamma))$

four terminal

Let's calculate γ_{11} first,

$$\gamma_{11} = \underset{\gamma}{\text{argmin}} \sum_{x_i \in R_{11}} L(y_i, f_0(x_i) + \gamma)$$

↳ We have to consider those points which belong to terminal region R_{11} (we have two such points)

$$\gamma_{11} = \underset{\gamma}{\text{argmin}} \frac{1}{2} \sum_{i=1}^{2} (y_i - (f_0(x_i) + \gamma))^2$$

$$\frac{\partial L_{11}}{\partial \gamma} = \sum_{i=1}^{2} (y_i - (f_0(x_i) + \gamma) \cdot (-1)$$

To get the minimum value, we ~~replace~~ equate it to 0.

$$\Rightarrow -\sum_{i=1}^{2} (y_i - f_0(x_i) - \gamma) = 0$$

$$\Rightarrow y_1 - f_0(x_1) - \gamma + y_2 - f_0(x_2) - \gamma = 0$$

$$\Rightarrow 3 - 4.8 - \gamma + 3 - 4.8 - \gamma = 0$$

$$\Rightarrow 2\gamma = -3.6 \Rightarrow \gamma_{11} = -1.8$$

Similarly, $\gamma_{21} = \underset{\gamma}{\text{argmin}} \sum_{x_i \in R_{21}} L(y_i, f_0(x_i) + \gamma)$

$$\gamma_{21} = \underset{\gamma}{\text{argmin}} \frac{1}{2} (y_i - (f_0(x_i) + \gamma))^2$$

[∵ since only one point in terminal region 2 we dont need Σ]

$$\frac{\partial L}{\partial \gamma} = \frac{1}{2} \times 2 [y_i - (f_0(x_i) + \gamma)] \cdot \frac{\partial}{\partial \gamma} [y_i - (f_0(x_i) + \gamma)]$$

$$\frac{\partial L}{\partial \gamma} = [y_i - f_0(x_i) - \gamma] \cdot (-1)$$

$$\Rightarrow y_1 - f_0(x_1) - \gamma = 0$$

$$\Rightarrow \gamma = -f_0(x_1) + y_1$$

$$\Rightarrow \gamma_{21} = -4.8 + 4 = -0.8$$

Similarly, we can calculate $\gamma_{31} = 3.2$ & $\gamma_{41} = 1.2$

* If we notice, the values of $\gamma_{11}, \gamma_{21}, \gamma_{31}$ & γ_{41} is exactly same as the values given by our decision tree leaf nodes. This happens because loss function we are using is least squares, if it was some other loss f^n these values would have been different.

(d) Now, $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$

for us $m=1$, so

$f_1(x) = f_0(x) + dt_1$ — this term is output of our decision tree

3. $\hat{f}(x) = f_1(x)$

i.e., output of first model $= f_0(x) + dt_1$.

but we also use a learning rate, let it be α or $\gamma = 0.1$.

$f_{pred_2} = f_1(x) = f_0(x) + \alpha dt_1$ $= m_1 + \alpha \times m_2$

$= 4.8 + 0.1 * (-1.8)$

$= 4.8 - 0.18$

$= 4.62$

Similarly, $4.8 + 0.1 * (-0.8)$

$= 4.8 - 0.08$

$= 4.72$

The $f_1(x)$ is output of model m_2

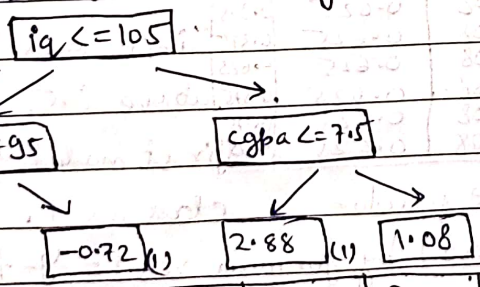
iq	cgpa	salary	$f_0(x)$	r_{i1}	$f_1(x)$
90	8	3	4.8	-1.8	4.62
100	7	4	4.8	-0.8	4.72
110	6	8	4.8	3.2	4.48
120	9	6	4.8	1.2	4.68
80	5	3	4.8	-1.8	4.62

Now, we calculate the residual for model 2, which will be —

iq	cgpa	salary	$f_0(x)$	r_{i1}	$f_1(x)$	r_{i2}
90	8	3	4.8	-1.8	4.62	-1.62
100	7	4	4.8	-0.8	4.72	-0.72
110	6	8	4.8	3.2	5.12	2.88
120	9	6	4.8	1.2	4.92	1.08
80	5	3	4.8	-1.8	4.62	-1.62

we can see for the second model, loss got reduced in each row.

Now, we need to repeat the entire process for model (m_3). When we train the decision tree using inputs and output as r_{i2} (residual of model 2) it will be like. —



iq	cgpa	salary	$F_0(x)$	r_{i1}	$f_1(x)$	r_{i2}	$f_2(x)$	r_{i3}
90	8	3	4.8	-1.8	4.62	-1.62	4.458	-1.458
100	7	4	4.8	-0.8	4.72	-0.72	4.648	-0.648
110	6	8	4.8	3.2	5.12	2.88	5.408	2.592
120	9	6	4.8	1.2	4.92	1.08	5.028	0.972
80	5	3	4.8	-1.8	4.62	-1.62	4.458	-1.458

Eventually, loss will decrease as we add more models.

Gradient Boosting for classification

Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations.

The loss function, we use for classification problem is log-loss.

We will use the same steps, we used for regression, one change is we are using different loss function.

Let's start with our dummy dataset –

cgpa	iq	is placed	pre1 (log-odds)
6.82	118	0	0.5108
6.36	125	1	0.5108
5.39	99	1	0.5108
5.50	106	1	0.5108
6.39	148	0	0.5108
9.13	148	1	0.5108
7.17	147	1	0.5108
7.72	72	0	0.5108

For this example, we will again be using combination of three models –

$f_0(x)$ $f_1(x)$ $f_2(x)$

$m_1 \rightarrow m_2 \rightarrow m_3 \rightarrow$

In regression, we saw the first model we use is a simple model which predicted the mean of target column, but in case of classification the first model will simply predict log-odds i.e., $\log\left(\frac{p}{1-p}\right)$. Therefore – log-odds = $\log_e\left(\frac{5}{3}\right)$ = 0.5108

Step 1: $\Rightarrow \log_e\left(\frac{5/8}{1-5/8}\right) \Rightarrow \log_e\left(\frac{5/8}{3/8}\right) \Rightarrow \log_e\left(\frac{5}{3}\right)$

Step 2: Now, we need to get probability (p) for each log-odds, which is calculated using the formula $\Rightarrow p = \frac{1}{1+e^{-\text{log-odds}}}$

cgpa	iq	is placed	pre1 (log-odds)	pre1 (probability)	res1
6.82	118	0	0.5108	0.625	-0.625
6.36	125	1	0.5108	0.625	0.375
5.39	99	1	0.5108	0.625	0.375
5.50	106	1	0.5108	0.625	0.375
6.39	148	0	0.5108	0.625	-0.625
9.13	148	1	0.5108	0.625	0.375
7.17	147	1	0.5108	0.625	0.375
7.72	72	0	0.5108	0.625	-0.625

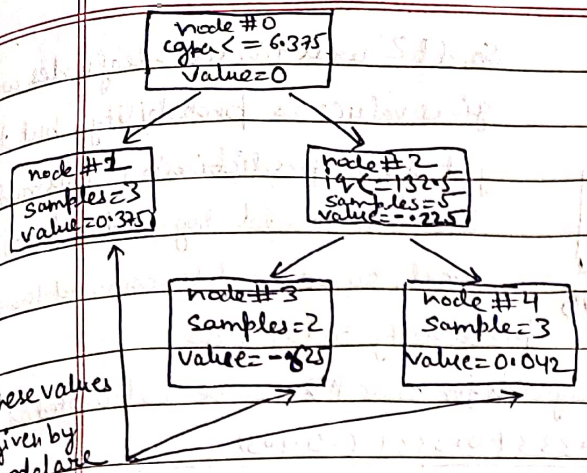
We will consider threshold as 0.5, since, our probabilities are above 0.5, we can say our first model, $f_0(x)$ predicts each class as 1.

Step 2: Now, we calculate pseudo-residual for model 1 (res1) by formula $res1 = y - f_0(x)$

We will now train a regression decision tree using input columns and res 1 as the output column, as we were doing in case of regression also.

When we train a regression tree (weak learner having max-leaf-nodes = 3), we get the tree as –

dt 1



In case of regression, we were directly taking these values as the prediction, but prediction for model 2, $f_1(x) = f_0(x) + \alpha dt_1$. Here $f_0(x)$ is log-odds and dt_1 produces probability, so they both can't be added directly, hence we need to convert probabilities to log-odds again. This is done using—

$$\frac{\Sigma \text{ Residual}}{\Sigma [\text{Prev. Prob} * (1 - \text{Prev. Prob.})]}$$

For using this formula, we need to know which sample was distributed in which terminal region/node.

cgpa	iq	is placed	pred 1 (log-odds)	pred 1 probability	res 1	node/ leaf	log-odds dt 1	pred 2 log-odds	pred 2 probab.
6.82	118	0	0.5108	0.625	-.625	3	-2.66	-2.159	0.103
6.36	125	1	0.5108	0.625	.375	1	1.6	2.110	0.891
5.39	99	1	0.5108	0.625	.375	1	1.6	2.110	0.891
5.50	106	1	0.5108	0.625	.375	1	1.6	2.110	0.891
6.39	148	0	0.5108	0.625	-.625	4	0.18	0.690	0.666
9.13	148	1	0.5108	0.625	.375	4	0.18	0.690	0.666
7.17	147	1	0.5108	0.625	.375	4	0.18	0.690	0.666
7.72	72	0	0.5108	0.625	-.625	3	-2.66	-2.159	0.103

For node 3, let's calculate log-odds

$$\frac{-0.625 - 0.625}{0.625(1-0.625) + 0.625(1-0.625)} = \frac{-2 \times 0.625}{2 \times 0.625 \times 0.375} = -2.659$$

Similarly, it can be calculated for node 1 & 4 as well.

Now, we calculate log-odds pred2 as $f_0(x)$ + log-odds (dt1)

$\Rightarrow$ $0.5108 + (-2.66) = -2.15$

$\Rightarrow$ $0.5108 + 1.6 = 2.11$ and so on. Now, we need to get probability again as shown above using pred2 log-odds.

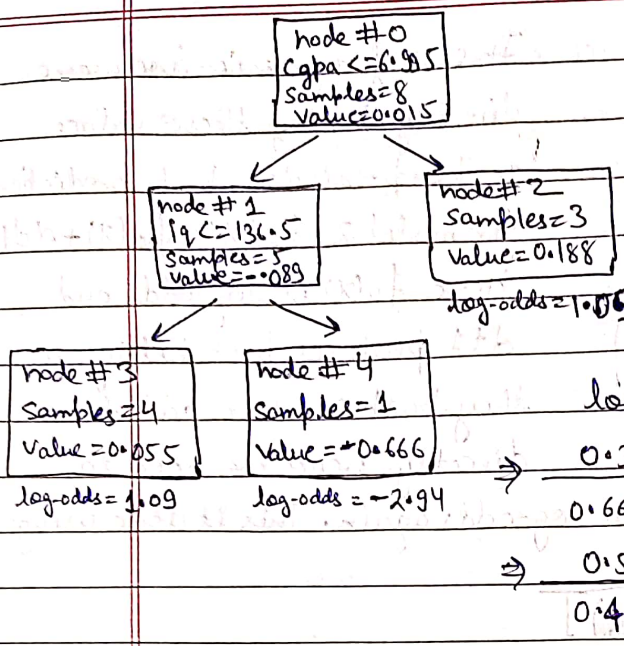
$\Rightarrow$ $\frac{1}{1+e^{+2.15}} = 0.103$, $\frac{1}{1+e^{-2.11}} = 0.891$ and so on.

This pred 2 is the prediction of our model 2 i.e., $f_1(x)$.

Step3: Now, we again calculate residual for $f_1(x)$ and then fit a decision tree using inputs and res2 as target.

cgpa	iq	is placed	pred 1 log-odds	pred 1 prob	res 1	log-odds dt 1	pred 2 log-odds	pred 2 prob	res 2
6.82	118	0	0.5108	0.625	-.625	-2.66	-2.159	.103	-.103
6.36	125	1	0.5108	.625	.375	1.6	2.110	.891	.108
5.39	99	1	0.5108	.625	.375	1.6	2.110	.891	.108
5.50	106	1	0.5108	.625	.375	1.6	2.110	.891	.108
6.39	148	0	0.5108	.625	-.625	0.18	0.690	.666	-0.666
9.13	148	1	0.5108	.625	.375	0.18	0.690	.666	0.333
7.17	147	1	0.5108	.625	.375	0.18	0.690	.666	0.333
7.72	72	0	0.5108	.625	-.625	-2.66	-2.159	.103	-0.103

dt2



So, dt2 will give us 3 leaf nodes which gives values as probability, but to get final prediction by this model we need to add log-odd of $f_1(x)$ as well, so we need to convert these probability to log-odds.

log-odds for node #2 can be calculated as -

$$\Rightarrow \frac{0.333 + 0.333 + (-0.103)}{0.666(1-0.666) + 0.666(1-0.666) + 0.103*(1-0.103)}$$

$$\Rightarrow \frac{0.563}{0.444 + 0.092} \Rightarrow \frac{0.563}{0.536} \Rightarrow 1.05$$

After getting log-odds for each node, we add $f_1(x) + dt2$ to get predicted log-odds for $f_2(x)$. Finally, we convert that log-odd to probability again giving us the final output.

Intp-,

Ques. Why residuals are called pseudo-residuals in gradient boosting?

Ans→ In context of gradient boosting, the term pseudo-residuals is used instead of just "residuals" because these residuals are not always the simple differences between actual value (target) and the predicted value. Instead, they represent the negative gradient of the loss function with respect to predictions at each iterations.

* In context of regression with MSE (Mean Squarred Error), the pseudo-residuals are indeed the same as the true residuals, because the gradient of MSE with respect to the predictions is just the difference between the predicted and actual values. However, in other cases, (e.g. classification using log loss), the gradients are more complex equation and don't correspond directly to simple residuals.

* By using pseudo-residuals, gradient boosting generalizes the concept of residuals to any differentiable loss function. This allows the algorithm to be applied to a wide variety of

problems, including classification and regression. This gradient based approach enables gradient boosting to work with a wide range of loss functions, making it a very versatile and powerful algorithm.

Ques. Differences between AdaBoost and Gradient Boost?

Ans→

AdaBoost	Gradient Boost
1) It focuses on adjusting sample weights based on errors made by previous model.	1) It focuses on minimizing a loss function by iteratively adding weak learners.
2) They typically use decision stumps which can have at most 1 split and 2 leaves.	2) They typically use decision tree of variable depth and can have more than 2 leaves.
3) Implicitly minimizes an exponential loss function.	3) Explicitly minimizes a customizable loss fn like (MSE, log-loss, etc.)
4) Re-weights mis-classified samples to focus on hard cases.	4) Fits new model to residuals of previous model without re-weighting.
5) No explicit learning rate, model weights adjust influence.	5) Explicit learning rate controls the contribution of each learner which is same for all.
6) More sensitive to outliers and prone to overfitting, since it aggressively focuses on mis-classified samples.	6) It is more robust, but can overfit when building weak learners which are fully grown decision trees.

Ques Compute the pseudo-residual i.e., negative residual of loss function in case of regression taking MSE and classification using log-loss / cross-entropy loss.

Ans→ We know that MSE is given as—

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_{act} - y_{pred})^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

& log-loss $= -\frac{1}{n}\sum_{i=1}^{n}[(y_i \log \hat{p}_i) + (1-y_i)\log(1-\hat{p}_i)]$

where $\hat{p}_i = \sigma(F(x_i)) = \frac{1}{1+e^{-F(x_i)}}$

* Negative gradient of MSE loss fⁿ for regression with respect to model's prediction $\hat{y}_i = F(x_i)$

$$\frac{\partial MSE}{\partial \hat{y}_i} = \frac{\partial}{\partial \hat{y}_i}\left(\frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2\right)$$

$$= \frac{1}{n} \times 2(y_i-\hat{y}_i)\cdot(-1)$$

{since the sum involves all n samples, the partial derivative w.r.t. $\hat{y}_i$ affects only the term corresponding to the i-th sample.}

$$\frac{\partial MSE}{\partial \hat{y}_i} = \frac{2}{n}(\hat{y}_i - y_i)$$

Now, negative gradient (pseudo-residual)

$$\text{negative gradient} = -\frac{\partial MSE}{\partial \hat{y}_i} = -\frac{2}{n}(\hat{y}_i - y_i) = \frac{2}{n}(y_i-\hat{y}$$

However, since Gradient boosting typically operates on a per sample basis, we can ommit $\frac{1}{n}$ factor and factor of 2 can be omitted as well since it is only for scaling.

Therefore, the pseudo-residual is typically expressed as -

$$\boxed{\text{pseudo-residual} = y_i - \hat{y}_i}$$

For a regression problem, using MSE as the loss fⁿ, the pseudo-residual is simply the difference between the actual value y_i and the predicted value $\hat{y}_i$.

* $$\frac{\partial \text{log-loss}}{\partial \hat{y}_i} = \frac{\partial \text{log-loss}}{\partial \hat{p}_i} * \frac{\partial \hat{p}_i}{\partial \hat{y}_i}$$ [Using chain rule]

$$\frac{\partial \text{log-loss}}{\partial \hat{p}_i} = \frac{\partial}{\partial \hat{p}_i} -\left[y_i \log(\hat{p}_i) + (1-y_i)\log(1-\hat{p}_i)\right]$$

$$= -\left[\frac{y_i}{\hat{p}_i} + \frac{(1-y_i)}{(1-\hat{p}_i)} * (-1)\right]$$

$$\frac{\partial \text{log-loss}}{\partial \hat{p}_i} = -\left(\frac{y_i}{\hat{p}_i} - \frac{1-y_i}{1-\hat{p}_i}\right)$$

Now, $$\frac{\partial \hat{p}_i}{\partial F(x_i)} = \frac{\partial}{\partial F(x_i)} \frac{1}{1+e^{-F(x_i)}}$$ [$\because \hat{y}_i = F(x_i)$]

$$= -\frac{e^{-F(x)}}{(1+e^{-F(x)})^2} = \hat{p}_i(1-\hat{p}_i)$$ [$\because \hat{p} = \frac{1}{1+e^{-F(x)}}$]

$$\therefore \frac{\partial \text{log-loss}}{\partial F(x_i)} = -\left(\frac{y_i}{\hat{p}_i} - \frac{1-y_i}{1-\hat{p}_i}\right) * \hat{p}_i(1-\hat{p}_i)$$

$$= -\left[\frac{y_i*(1-\hat{p}_i) - \hat{p}_i(1-y_i)}{\hat{p}_i*(1-\hat{p}_i)}\right] * \hat{p}_i(1-\hat{p}_i)$$

$$= -\left[y_i - y_i\hat{p}_i - \hat{p}_i + y_i\hat{p}_i \right]$$

$$\frac{\partial \text{logloss}}{\partial F(x_i)} = \hat{p}_i - y_i$$

Now negative of gradient of loss fⁿ, i.e.,

$$\text{pseudo residual} = -(\hat{p}_i - y_i) = y_i - \hat{p}_i$$

The negative gradient, $y_i - \hat{p}_i$ is what gradient boosting uses as the pseudo-residual at each iteration to fit the next decision tree. The pseudo-residual represents how much the model's predicted probability deviates from the actual label, guiding the next trees to correct these deviation