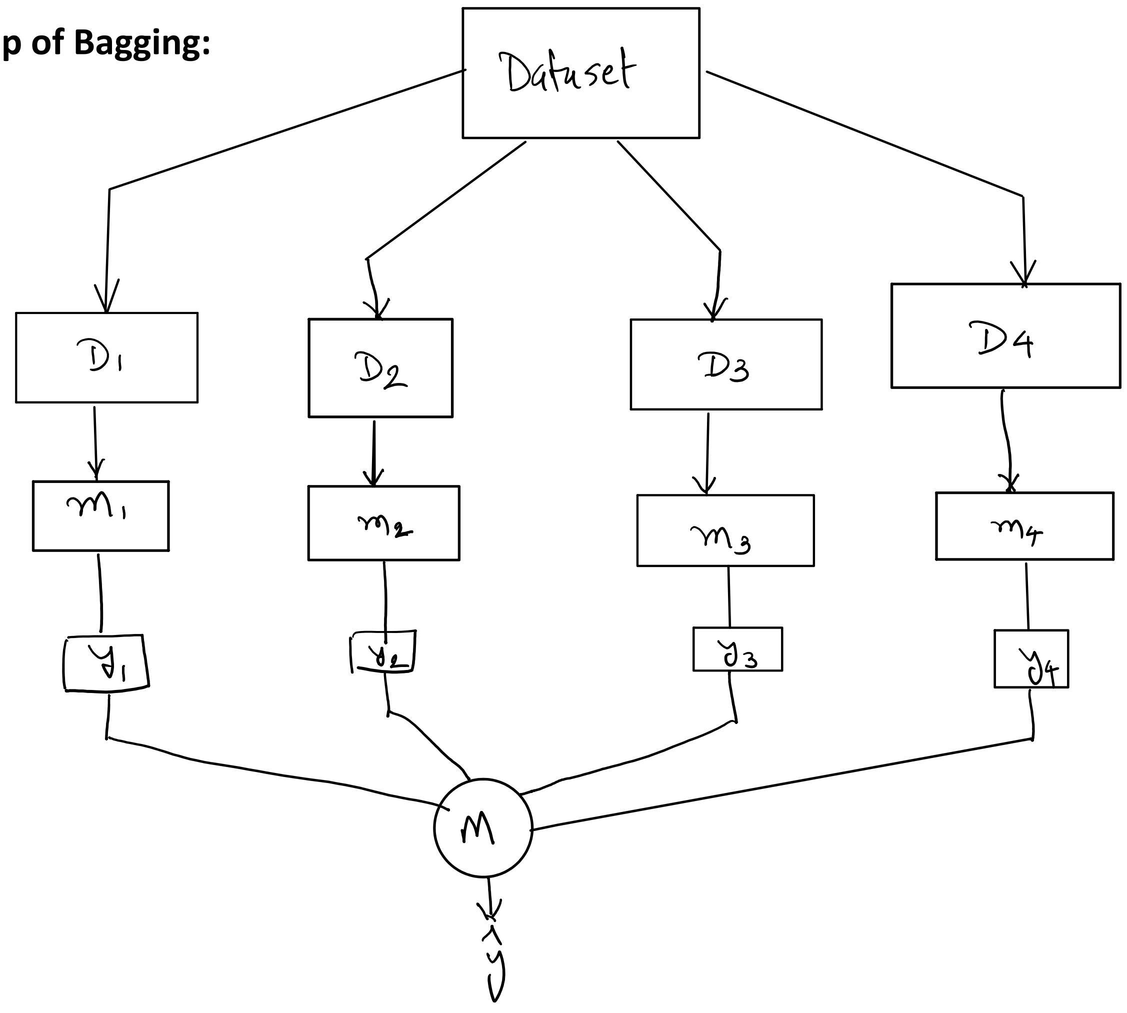
Boosting

10 April 2025 20:19

Recap of Bagging:



How Boosting is different from Bagging?

1. Our base learners will be underfitted in Boosting - how can we create underfit base learners?
    a. Using mean model
    b. Using a "decision stump"
2. Instead of parallelly providing the data to different base learners, we will feed the data sequentially to these learners.

Let's understand this concept with an example.

Let's take 4 students: Prince, Jay, Hardik, Mitanshu
We have a code of 20 lines but it throws errors.
We give this code to Prince. Prince checked & corrected first 5 lines of code then he was busy so he gave this code to Jay.
Now what Jay is going to work on? Jay will start working on the code from line 6. Suppose he corrected next 7 line and sent the code to Hardik.
From where Hardik will start? He will start from line 13. Let's say he also improved the next 6 lines and submitted that code to Mitanshu.
Now Mitanshu needs to do? Just the remaining 2 lines.

But how to implement this logically?

1. A single datapoint is taken and its label is predicted using "mean model"
2. Error in the prediction is calculated

$$\begin{tikzcd}
 & \hat{y}=12 & err_2=(-2) & err_3=(-1) & err_4=(-0.75) & err_5=(-0.15) \\
D \arrow[r] & \boxed{m_0} \arrow[r] & \boxed{m_1} \arrow[r] & \boxed{m_2} \arrow[r] & \boxed{m_3} \arrow[r] & \boxed{m_4} \\
 & (x_i, y_i) & (x_i, err_1) & (x_i, err_2) & (x_i, err_3) & (x_i, err_4)
\end{tikzcd}$$

$(x_i, y_i) \downarrow$ 8

Error = 4 = err₁ → Target Variable for m_1

$\boxed{\hat{y}_0 = 8}$

$\hat{y}_1 = \hat{y}_0 - E_2$
$= 8 - (-2)$
$= 8 + 2$
$\boxed{\hat{y}_1 = 10}$

$\hat{y}_1 - err_3$
$= 10 - (-1)$
$\boxed{\hat{y}_2 = 11}$

$\hat{y}_2 - err_4$
$= 11 - (-0.75)$
$\boxed{\hat{y}_3 = 11.75}$

$\hat{y}_3 - err_5$
$= 11.75 - (-0.15)$
$\boxed{\hat{y}_4 = 11.90}$

				m_0		m_1		m_2	
f1	f2	f3	Y	Y_0	E_0	Y_1	E_1	Y_2	E_2
			3	5.5	-2.5	-1.5	-1	-0.5	-0.5
			4	5.5	-1.5	-0.5	-1	-0.7	-0.3
			6	5.5	0.5	0.5	0	0	0
			9	5.5	3.5	2	1.5	0.5	1

mean of Y = 5.5
= O/P of mean model

Computation of errors:

$e_0 = y - \hat{y}_0$; y = Actual Label & $\hat{y}_0$ = Predicted label of mean model.

$e_1 = y - \hat{y}_0 - \hat{y}_1 \Rightarrow e_1 = y - (\hat{y}_0 + \hat{y}_1)$

$e_2 = y - \hat{y}_0 - \hat{y}_1 - \hat{y}_2 \Rightarrow e_2 = y - (\hat{y}_0 + \hat{y}_1 + \hat{y}_2)$

$$e_k = y - (\hat{y}_0 + \hat{y}_1 + \hat{y}_2 + \dots + \hat{y}_k)$$

$$e_k = y - \left[\hat{y}_0 + \sum_{i=1}^{k} y_i\right]$$

These models m₀, m₁, m₂, … are predicting errors and we want to minimize these errors. So, let's take the outputs of these models (errors) as a function of X (features) so we can treat that function as our loss function and then we can think of minimizing it.

$\therefore \hat{y}_0 = h_0(x),\ \hat{y}_1 = h_1(x),\ \hat{y}_2 = h_2(x), \ldots$

And let's say our final predictions are represented as $f_i(x)$ then they can be given as:

$$f_1(x) = h_0(x) + h_1(x)$$

$\therefore$ The step-1 is to create & train our first model m_1 on (x_i, e_0) & to get the above output.

step-2 : Create & train m_2 on (x_i, e_1) & to get the following o/p :

$$f_2(x) = h_0(x) + h_1(x) + h_2(x)$$

But just like we used to compute feature importance, here we want to find "model importance", let's also add weights to the o/p of each model. We will call these weights γ (gamma)

$$\boxed{f_1(x) = h_0(x) + \gamma_1 \cdot h_1(x)}$$

$$\boxed{f_2(x) = h_0(x) + \gamma_1 h_1(x) + \gamma_2 \cdot h_2(x)}$$

Continuing till step-k :

$$f_k(x) = h_0(x) + \gamma_1 h_1(x) + \gamma_2 h_2(x) + \ldots + \gamma_k h_k(x)$$

$$\boxed{\therefore f_k(x) = h_0(x) + \sum_{i=1}^{k} \gamma_i h_i(x)}$$

As we can observe, this formula is very similar to Linear Regression:

$$f_n(x) = h_0(x) + \gamma_1 h_1(x) + \gamma_2 h_2(x) + \ldots + \gamma_n h_n(x)$$

↳ Gradient Boosting

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

↳ Linear Regression

In L.R., w_i can be computed parallely but in G.B. we have to find γ_1 to calculate/find γ_2 so finding weights is a sequential process in Gradient Boosting

One another difference is that we can increase or decrease number of models from gradient boosting but we can not remove/add features in Linear Reg. hence no. of γ_i can change but not no. of w_i

Errors (Residuals) at kth stage:

$$\text{Residual} = y - f_k(x)$$

If we take "squared loss" (taking squares of the errors) as our loss function:

$$\text{squared loss} = (y - \hat{y})^2$$

$$L(y_1, \hat{y}_1) = (y_1 - \hat{y}_1)^2$$

$$L(y_i, f_k(x)) = (y - f_k(x))^2$$

$$L(y_i, f_k(x)) = (y - f_k(x))^2$$

Let's take $f_k(x)$ as z_i $\Rightarrow$ $z_i = f_k(x)$

$$L(y_i, z_i) = (y_i - z_i)^2$$

$$\frac{\partial L}{\partial z_i} = \frac{\partial}{\partial z_i}(y - z_i)^2$$

$$= 2(y_i - z_i) \cdot \frac{\partial}{\partial z_i}(-z_i)$$

$$\frac{\partial L}{\partial z_i} = -2(y_i - z_i)$$

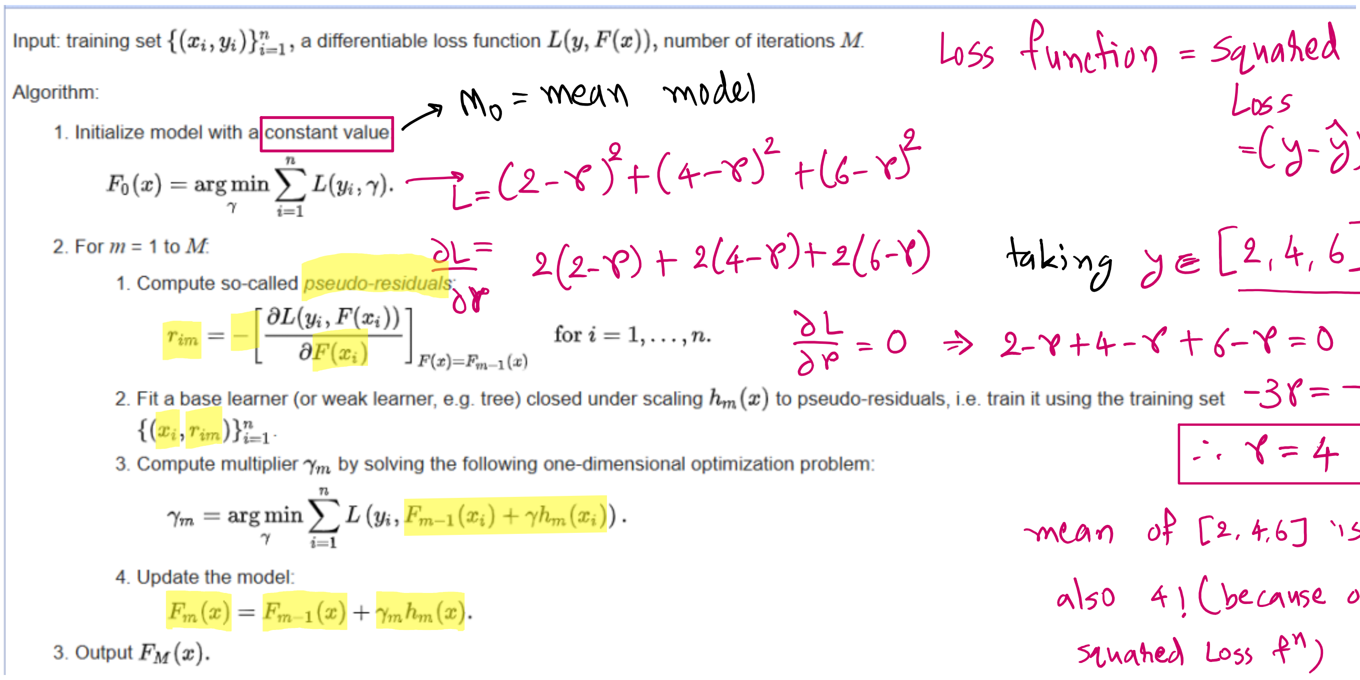
$$\frac{-\partial L}{\partial z_i} = \underbrace{2}_{\text{constant}}(y_i - z_i) \longrightarrow \text{Negative Of Gradient}$$

$$\boxed{-\frac{\partial L}{\partial z_i} \approx (y_i - z_i)}$$

<u>OR</u>

pseudo errors

$$\boxed{\frac{-\partial L}{\partial f_k(x)} = y_i - f_k(x_i)}$$



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

Algorithm:

1. Initialize model with a $\boxed{\text{constant value}}$: $\xrightarrow{}$ M_0 = mean model

   $$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma). \longrightarrow L = (2-\gamma)^2 + (4-\gamma)^2 + (6-\gamma)^2$$

2. For m = 1 to M:
   1. Compute so-called pseudo-residuals:

      $$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

   2. Fit a base learner (or weak learner, e.g. tree) closed under scaling $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
   3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

      $$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

   4. Update the model:

      $$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

Loss function = Squared Loss $= (y - \hat{y})^2$

$$\frac{\partial L}{\partial \gamma} = 2(2-\gamma) + 2(4-\gamma) + 2(6-\gamma)$$

taking $y \in [2, 4, 6]$

$$\frac{\partial L}{\partial \gamma} = 0 \Rightarrow 2-\gamma+4-\gamma+6-\gamma = 0$$

$$-3\gamma = -12$$

$$\boxed{\therefore \gamma = 4}$$

mean of [2, 4, 6] is also 4! (because of squared Loss f^n)

Hyper-parameter tuning:

Hyper parameter - 1: Number of base learners (models) Let's call it *'m'*.

What will happen if *m* is too large?

- Error values will be minimized (tend to 0)
- Final Model Overfits

What will happen if *m* is too small?

- Output will be close to mean value
- Final Model Underfits

Hyper parameter - 2: Depth of each base learner. Let's call it *'d'*.

What will happen if *d* is too large?

- Each base learner will overfit

What will happen if *d* is too small?

- Each base learner will underfit - preferred in Boosting

Regularizer/Regularization:

$$f_m(x) = h_0(x) + \underbrace{\nu}_{} \sum_{i=1}^{m} \gamma_i h_i(x)$$

$\hookrightarrow$ Regularizer| Regularization constant $[0 \leq \nu < \infty]$

If $\nu = 0 \Rightarrow f_m(x) = h_0(x)$

If $\nu = 0 \Rightarrow f_m(x) = \underbrace{h_0(x)}$ $\rightarrow$ o/p of mean model

$\rightarrow$ Underfit

as $\nu \rightarrow \infty \Rightarrow f_m(x) = \underbrace{h_0(x)}_{\text{negligible}} + \underbrace{\nu \sum_{i=1}^{m} \gamma_i h_i(x)}_{\text{dominant}}$

Problems with GBDT (Gradient Boosted Decision Tree): GBDT = pseudo residuals + Additive Combining

- Slow
- Sequential
- It starts overfitting very quickly

Impact of Outliers:

y	$\hat{y}_0$	err
2	37	-35
4	37	-33
6	37	-31
4	37	-33
6	37	-31
200	37	+163

& squared loss will further worse the situation

$\boxed{\therefore \text{ Impact of Outliers is Huge}}$

Model will focus on the error coming from the outlier only!!

Other Ensemble Techniques:

1. **XGBOOST**
2. **Light GBM**
3. **Stacking**
4. **Cascading**

XGBOOST:

Why?

Because GBDT is slow in training & very slow if we do hyperparameter tuning

Motivation:

Reduce the training time

How?

Optimizing training process

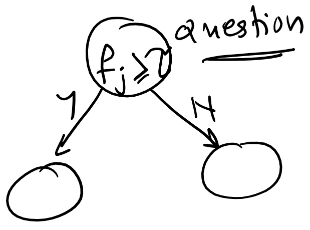
How a typical GBDT works:

$h_0(x)$ = mean model

$h_1(x)$: Decision Tree with very small depth.

→ How will we create this?

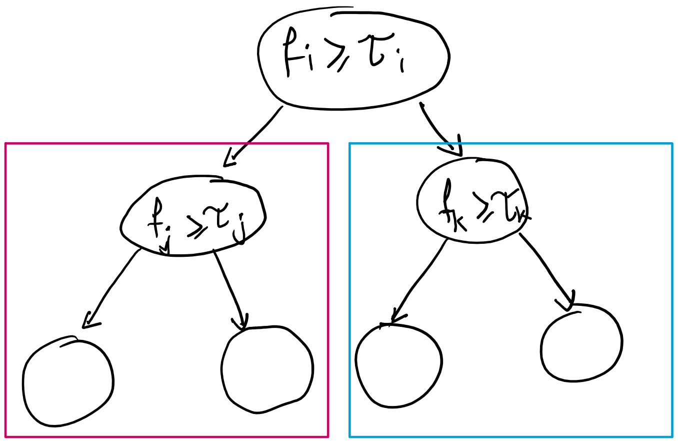
<u>Ans</u> - We calculate IG of every feature & then choose the feature with max. I.G.



Hence, rather than considering all features let's randomly choose a few of them only and train our model on those features. This is one way (column sampling/feature selection)

Moreover, we can also do the feature selection parallelly in order to calculate IG.

2nd Idea: Parallelization in building subtrees



Although the o/p of $h_2(x)$ can't be computed parallely with o/p of $h_1(x)$, we can parallely create the sub-trees of $h_1(x)$ or that of $h_2(x)$.

3rd Idea: Creating optimal bins on the numerical features to reduce number of thresholds for numerical columns

Numerical Col	bin
3	1 - 10
7	1 - 10
15	11 - 20
18	11 - 20
5	1 - 10

Normal way (without using bins) - We have to compute IG for 6 unique values.

<u>with bins</u> - Needs IG to be computed only for 2 bins i.e., 1-10 & 11-20

15	11 - 20
18	11 - 20
5	1 - 10
19	11 - 20

with bins – Needs [illegible]

for 2 bins . i.e., 1-10 & 11 - 20

Optimal bins – Ⓐ 1-5, 5-10, 10-15, 15-20 OR

→ Ⓑ 1-10, 11-20 OR

× Ⓒ 1-20, 21-40

2. Light GBM

Uses GOSS - Gradient-based One Sided Sampling

Cascading

We use cascading when the margin of error is extremely low or in other words, we cannot afford mistakes.

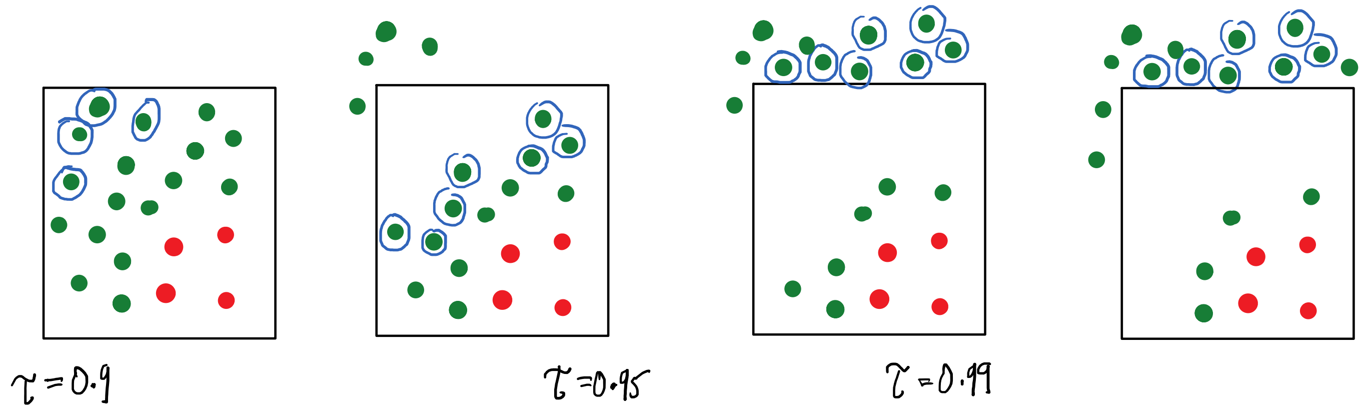
e.g.,
Medical field: classifying a malignant tumors and benign tumors
Finance sector: fraudulent vs. genuine transactions

Suppose we have the data of several transactions as below:

$$\begin{tikzcd}
 & \text{990 genuine} \\
\text{1000 transactions} \arrow[ur] \arrow[dr] & \\
 & \text{10 fraudulent}
\end{tikzcd}$$

Clearly the data is extremely imbalanced. What can we do to it?
Options: SMOTE?
What do we do in SMOTE?
SMOTE will create 980 "artificial/generated" transactions which will be labelled as "fraudulent" (by SMOTE) to balance the data. So at the end we will have 1980 data points in which 980 (nearly half of them) are synthetic!
Can we expect a model that is train on this data not to get wrong?

Then how can we solve this problem?
Solution: Visualize the bucket of data as shown in the figure. If we create a model that can predict probability of a particular transaction of being in majority class (being genuine), it will predict different probabilities for different transaction. Now let's decide a threshold (usually very high) e.g., 90%



In terms of ML language:

$$\begin{tikzcd}
\mathcal{D} \arrow[r] & M1 \arrow[r, "D-D_1'"] \arrow[d, "\boxed{P(y=0|\mathcal{D})}\ P(y=1|\mathcal{D})"', "\geq 90\%"] & M2 \arrow[r, "D-D_1'-D_2'"] \arrow[d, "{P(y=0|D-D_1') \geq 95\%}"] & M3 \arrow[r] & \cdots\cdots \arrow[r] & \text{Human Varification} \\
 & D_1' & D_2' & & &
\end{tikzcd}$$