Ensemble Models

☆ Group of musicians performing together

In Machine learning → multiple models used together

$$\underbrace{\{ M_1, M_2, M_3, \ldots, M_k \}}_{\downarrow} \rightarrow \text{base models}$$

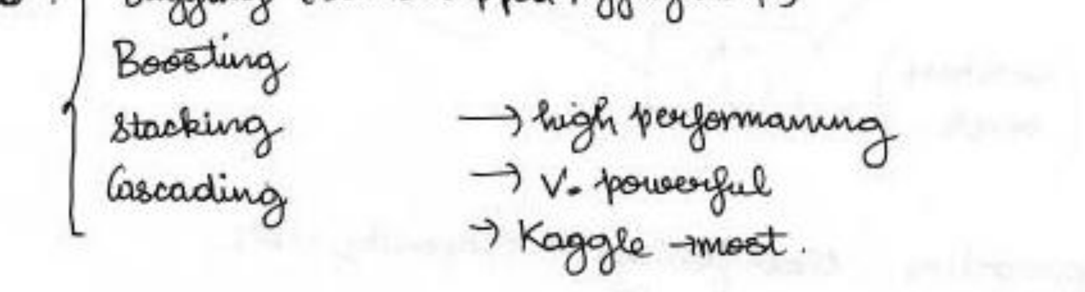
(Combine) more "powerful" model

4 Types ÷
- Bagging (Bootstrapped Aggregation)
- Boosting
- Stacking → high performing
- Cascading → V. powerful → Kaggle most.

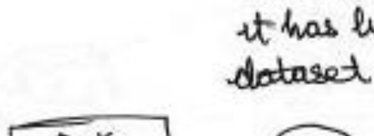
1) Bagging (Bootstrap Aggregation)

↳ statistics (Taking new sample from one sample)

Intuition : Random Forest is one of the most popular model used for bagging.

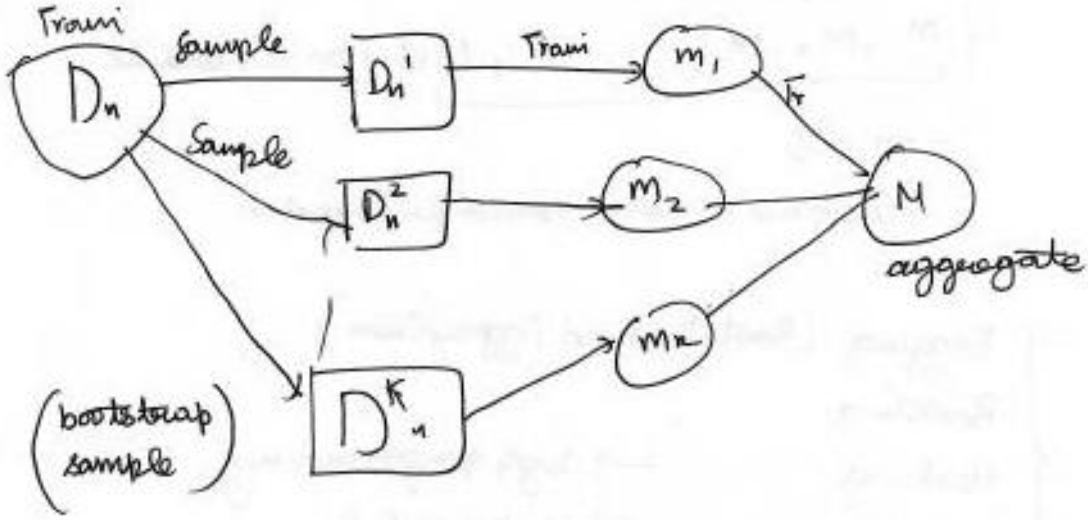


NOTE: Sampling with replacement → taken the point from dataset but it remains in dataset from which it has been taken as well as original dataset



M_i is built using D_n^i of size m $(m \le n)$

⇒ Each model M_i has seen a different subset of data.

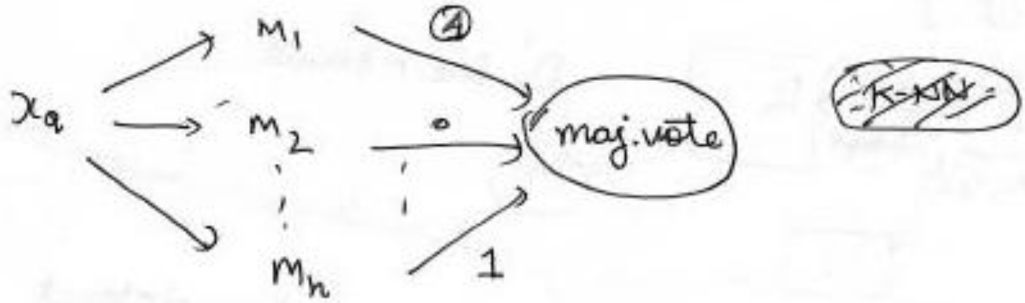


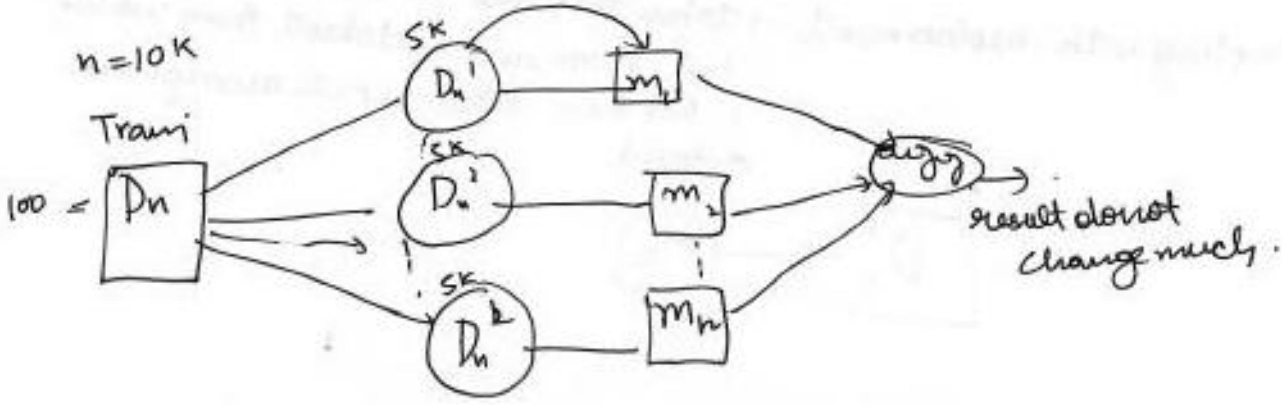
Aggregation : 1) Classification → Majority vote

2) Regression → mean/median

⇒ After training this model

I get $M_1, M_2 \dots M_k$ we get majority vote.





Limitation

1) model changes a lot ~~when~~ with change in ~~data~~dataset (Variances)

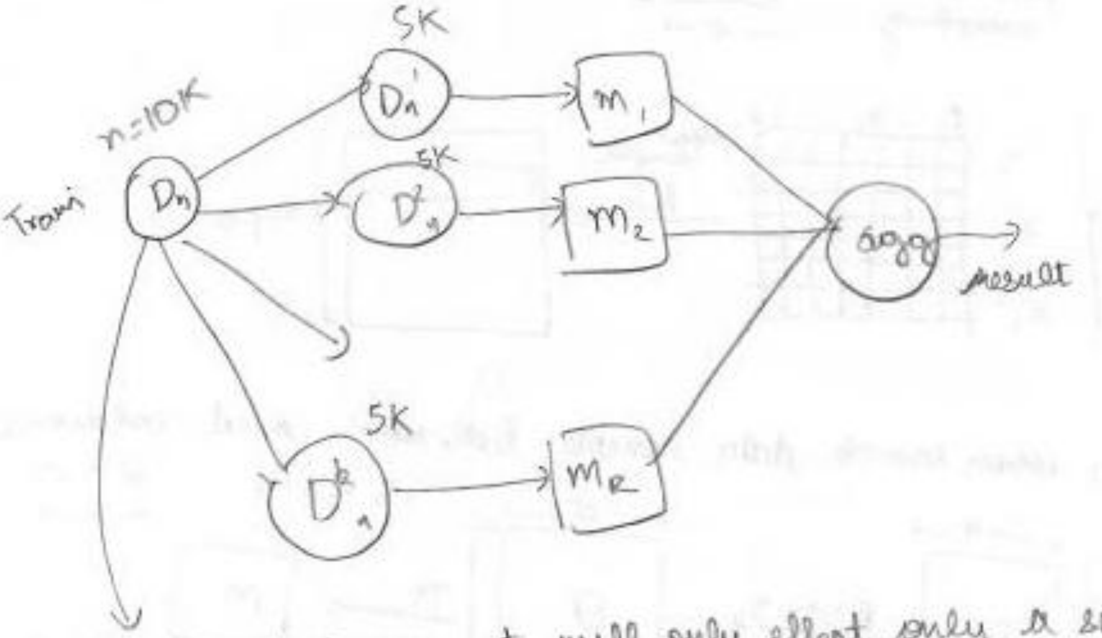
Bagging → can reduce variance in a model without impacting the bias.

model error = $Bias^2 + Var$

Base-model (M_i) ; - low bias, high var model

Bagging (M_i's) → low bias ; reduced variances.

Variance → {model changes a lot with change in D_n}



⇒ If dataset changes it will only effect only a small subset of models which will not affect the aggregate function a lot or by removing a bunch of points from D_n (train dataset) model donot change much.

NOTE: Bagging takes ~~bunch~~ bunch of low-bias, high variance models and combine them using bagging and convert them into low bias and low + reduced variance models.

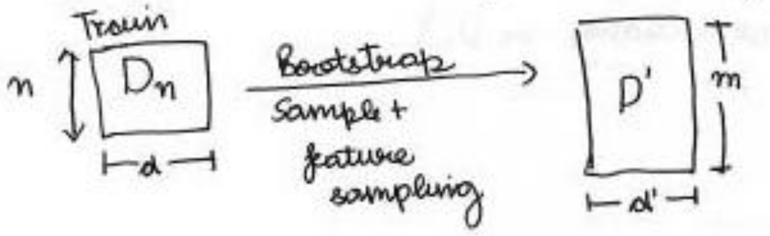
Random Forest (Bagging Technique Used)

Decision Trees ÷ As decision tree which have large depth has high variance (overfitting) and lower bias it is used to bagging technique to reduce variance and that technique is called Random Bootstrap Sampling

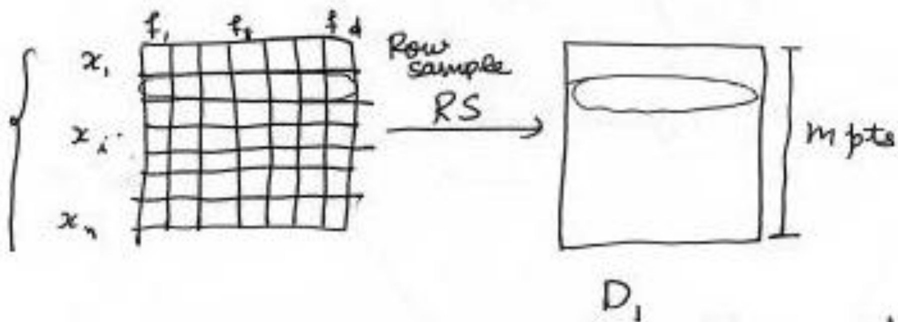
$$RF : D.T \uparrow \text{Bagging} + \underbrace{\text{Column Sampling}}_{\text{feature bagging}}$$

D.T ↑ base

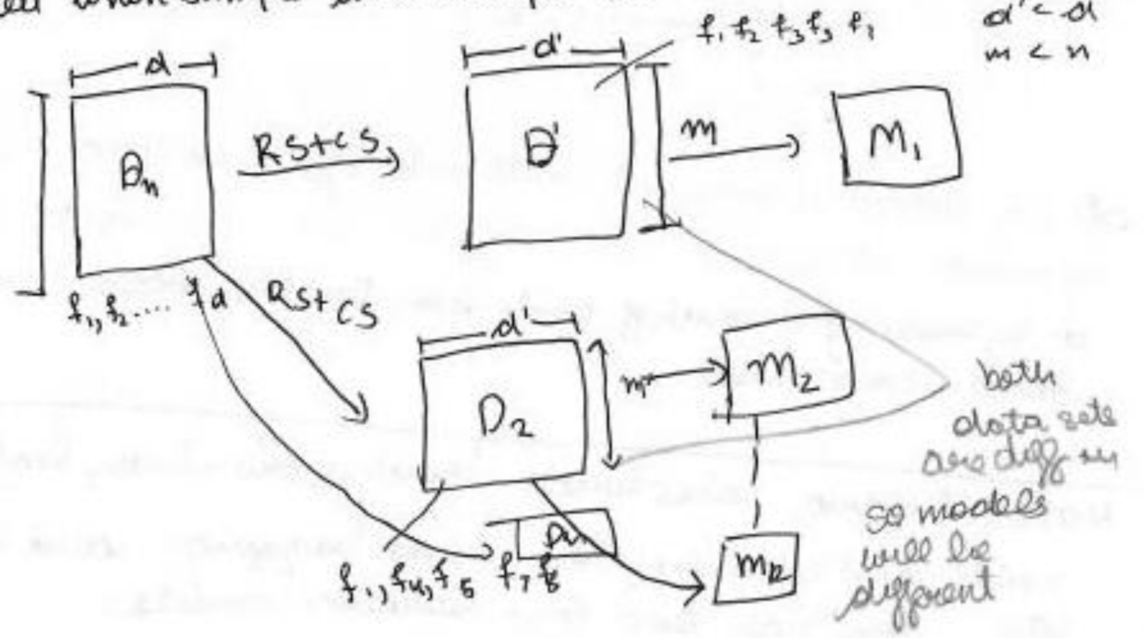
Bagging ↑ (Row sampling with replacement)



$$\begin{cases} m < n \\ d' < d \end{cases}$$



Core idea when sample data sample both row and column



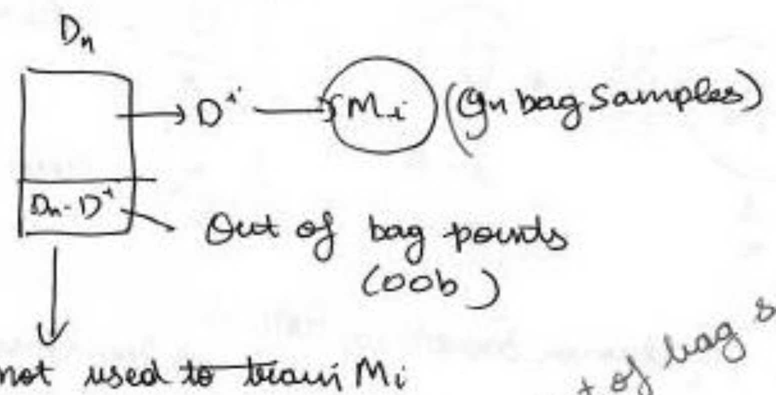
$M_1, M_2, \ldots, M_k$ are totally different

so after that we take the majority among $M_1, M_2, \ldots, M_k$.

low bias ; high variance

$M_i \longrightarrow D_i \longrightarrow d'$ cols, m rows $\quad (d' < d)$

$\boxed{D_n - D^i}$



since model M_i using point in D_i — out of bag samples

$\boxed{D_n - D_i \rightarrow \text{oob samples} = \text{CV-dataset}}$

RF : DT base learners of reasonable depth.

+ doing row sampling with replacement
+ column sampling
+ agg. (majority vote in case of classification)
  (mean or median in case of regression)

Bias - variance tradeoff

RF → reduce ~~variable~~ variance due to bagging

→ low bias because base learners (M_i's) are low bias

→ var

$M = agg(\underbrace{M_1, M_2, M_3}, \ldots, M_k)$ → These all are high depth Decision Tree having high variance same bias.

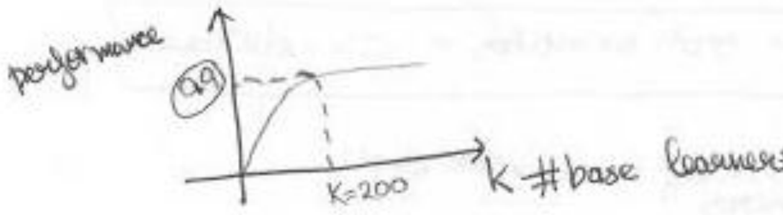
bias (m) = bias (individual model) $\begin{cases} K\uparrow, \text{ variance} \downarrow \\ K\downarrow, \text{ variance} \uparrow \end{cases}$

$$\underset{\substack{d \\ n}}{D_n} \xrightarrow[CS]{RS} \underset{\substack{d' < d \\ m < n}}{D'}$$

$\left\{ \dfrac{d'}{d} \right.$ = Column Sampling Ratio

$\left\{ \dfrac{m}{n} \right.$ = row Sampling Ratio

$\left\{ \begin{array}{l} \text{If column Sampling Rate} \downarrow \\ \text{Row Sampling Rate} \downarrow \end{array} \right.$ → low variance model Var ↓

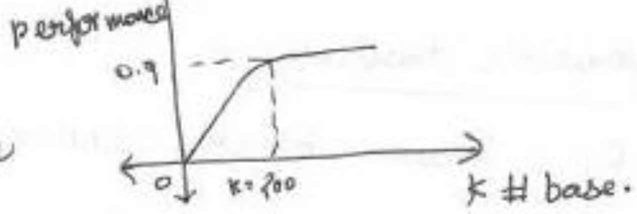


Only variance reduces but bias remains the same that is bias of trained model M remains same as the bias of individual mode M_i

D_v →

D_{Test} →

↓ report your model



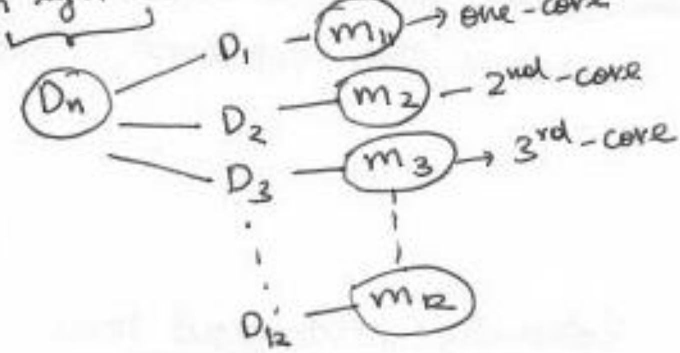
→ most important hyperparameter here is k. That is number of samples .. (number of base learners)

Train and Run Time Complexity.

RF with K base-learners (Decision Tree)

multi-core → system → we can train each model independently parallely.

Train: $O(\underbrace{n \lg n\, d}_{} * K)$

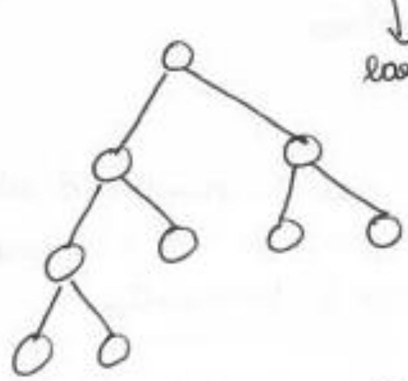


Trivially parallelizable

When large amount of data with reasonable number of features (d) you can train 1000 of trees very fast.

$\boxed{\text{peta bytes of data} \rightarrow \text{Train models}}$

Run time :- $O(\underbrace{depth}_{\text{large (10 to 20)}} * K)$



Space : $O(DT * K)$

Decision Tree :- with 200 Trees of depth = 6

Code :

sklearn. ensemble. RandomForestClassifier

Class sklearn. ensemble. RandomForestClassifier ($\overbrace{\text{n\_estimator}}^{K}$ = 10, criterion = 'gini', max_depth = None, min_samples_split = 2, min_samples_leaf = 1, min_weight_fraction_leaf = 0.0, max_features = 'auto', max_leaf_nodes = None, min_impurity_decreases = 0.0, min_impurity_split = None; bootstrap = True, oob_score = False, ...

Extremely Randomized trees

Regular decision trees ← RF. → Column Sampling ; Row sampling ; aggregation

Extreme Trees : → M_i →

f_i : real valued feature than we can put threshold on feature i.

RF / Decision Tree

- f_i : real valued values
- sort f_i

1 ← τ
2 ← τ
3 ← τ
4 ← τ
5 ← τ
6

pick the threshold which give me the maximum gini impurity

NOTE: They say instead trying threshold on all values Try out on random ~~funct~~ sample of possible values.

Extreme Trees : col. sampling + Row sampling + aggregation + $\underbrace{\text{randomization when selecting } \tau}$

This is the part different from Random forest.

Radomization as a way to reduce variance.

R.F → C.Sampling + Row.Sampling

Extremely Randomized trees → Column Samp. + Row Sampling + random "τ"

Threshold.

They reduce variance better than Random forest but bias might increase.

Random Forest - Cases

RF : Decision Tree + $\overbrace{\text{Row Samp + Column Samp + aggregation}}^{\text{reduce variance}}$

↳ large-dim ; Cat - features with many categories

Cases for DT

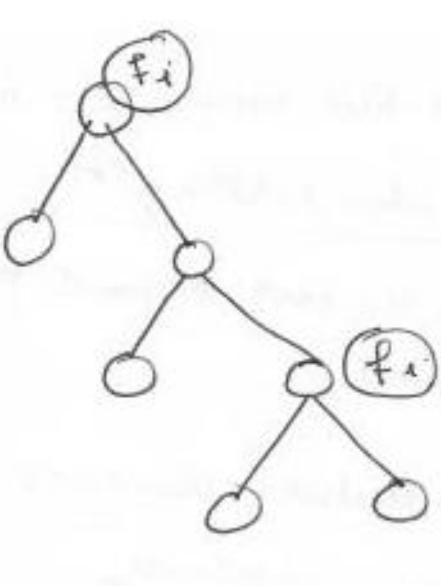
Decision Tree → depth

Bias-Variance of Random forest ~ K = # of base learners.

↳ deep trees Train but control variance.

(2) Feature Importance →

DT ÷ f_i :- overall reduction in Entropy or Gini Impurity because of this feature @ various levels in the Decision Tree



RF → K Decision Tree → $M_1, M_2, \dots, M_k$

RF → K Decision Tree

overall reduction in entropy ~~and~~ or Info gain because of f_i @ various levels of each of M_i's

If feature used in lot of decision tree so overall importance of feature increases

☆ $\boxed{\text{Where decision tree work well Random forest also work well and vice verse. (but not in case of bias variance tradeoffs)}}$

Boosting :

Bagging → high variance & low bias model + randomization (RS; CS) + aggregation — to reduce variance.

Boosting ⇒ $\underbrace{\overbrace{\text{low variance, high bias model}}^{\text{reduce the bias here}}}_{}$ + $\underbrace{\text{additively combining}}_{}$

reduce bias.
while keep our var. low

$$\text{Error} = (\text{bias})^2 + \text{Var} + \underset{\text{irreducible error}}{\varepsilon}$$

Thes NOTES FOR BAGGING:

NOTE :
- Reduces variance of the base learners.
- Bootstraps the training data and train learners in parallel
- Each learner is often trained on a random subset of training data
- learners vote on the outcome with weights

eg: DT which is shallow not too deep

Core Idea :- reduces bias

$$D_{Train} = \{x_i, y_i\}_{i=1}^{n}$$

① D_{Train} $\{x_i, y_i\}_{i=1}^{n}$ → M_0 → high bias, low variance

$\boxed{\text{high-bias} \rightarrow \text{Underfitting or large training error.}}$

(a) D_{Train} $\{x_i, y_i\}_{i=1}^{n}$ → M_0 → high bias ⇒ large Train error.

$y = h_0(x)$

(b.) $y_i - h_\theta(x_i) = error_i$ in pt type ← eg { squared error, log error, huge error.

simple diff error.

$$\{x_i, y_i, error_i\}_{i=1}^{n}$$

① $M_1 \leftarrow \underbrace{\{x_i, \widetilde{error_i}\}_{i=1}^{n}}$.

$h_1(x)$

Iteration 1 $F_1(x)$ = model at end of stage 1 / Iteration 1

$$F_1(x) = \alpha_0 h_0(x) + \alpha_1 h_1(x)$$

Iteration ② $\{x_i, error\}$ → $y_i - F_1(x_i)$

$$F_2(x) = \alpha_0 h_0(x) + \alpha_2 h_2(x) + \alpha_1 h(x)$$

end of stage k :-

$$F_K(x) = \underbrace{\sum_{i=0}^{K} \alpha_i \overbrace{h_i(x)}^{\text{trained to fit the residual error.}}}_{\text{additive weighted model}}$$

$h_i(x) \leftarrow \{x_i, err_i\}$

err_i : residual error @ end of stage $(i-1)$ $\rightarrow y_i - F_{i-1}(x)$

* Final model

$K = 10$

$$F_K(x) = \sum_{i=0}^{K} \alpha_i \underbrace{h_i(x)}_{\text{residual error}}$$

ends up having.

a low residual error

Training error reduces

After every iteration residual error reduces.

→ Gradient Boosted Decision Tree (GBDT)

→ Adaptive Boosting → Images (Face detection algorithm)

BOOSTING NOTES

Reduces bias of the bias learners.

Build learners sequentially.

Sample misclassified by previous learners weighted more in subsequent learners

Residuals, loss-functions & Gradients

$$F_k(x) = \sum_{i=0}^{K} \underbrace{\alpha_i}\ \underbrace{h_i(x)}$$

$M_{K+1} \leftarrow \{x_i, err_i\}$

residual @ end of stage K $\Big\{$ $err_i = y_i - F_k(x)$

$\uparrow$ Residual

whenever we solve machine learning problem

↓

we try to minimize loss function

for Square - loss

$$\mathcal{L}(y_i, F_K(x_i)) = (y_i - F_k(x_i))^2$$

$\Big\{$ Let $F_K(x_i) = Z_i$

SQ - loss $\dfrac{\partial L}{\partial F_K(x_i)} = \dfrac{\partial L}{\partial Z_i} = \dfrac{\partial}{\partial Z_i}(y_i - Z_i)^2$

$$= (-1) * 2 * (y_i - Z_i)$$

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

$$\underbrace{-\frac{\partial \mathcal{L}}{\partial F_k(x_i)}} = 2\,(\underbrace{\overbrace{y_i - F_K(x_i)}^{err_i}}_{\text{Residual}})$$

-ve derivative of loss function w.r.t model at stage K.

(*) negative gradient ≈ residual

$$\boxed{\text{Gradient boosting is so powerful due to pseudo-residual. that is negative gradient}}$$

$$h_i(x) \leftarrow (x_i, err_i) = y_i - F_{i-1}(x)$$

$err_i \rightarrow$ residual;

$err_i \leftarrow$ pseudo-residual

① negative gradient ≈ residual

↑ pseudo-residual

* Why training on pseudo-residual but why not residual?

pseudo-residual has an advantage that is any it works on any loss function which is differentiable

for eg $\boxed{\text{RF} \rightarrow \text{min hinge loss}}$ → not possible

but with
Gradient Boosting → min (any loss) → differentiable

↑ super powerful

Stage: i $F_{i-1}(x)$

↓ train

$(x_i, err_i) \longrightarrow M_i$ → model

$\boxed{\dfrac{-\partial L}{\partial F_{i-1}(x)}}$ ← pseudo residual

$h_i(x)$

Gradient Boosting → If you have base model having ~~low~~ high bias → combine pseudo-residual

$\hookrightarrow \frac{\partial L}{\partial F_{i-1}(x)}$

Pseudoresidual for log-loss and classification

$\{ -\frac{\partial L}{\partial F_K(x)} \approx$ residual/errors for nm squared-loss functions.

regression $\frac{-\partial L}{\partial F_K(x)} = 2(y_i - F_K(x_i))$

Binary Classification task → Using log-loss

$\frac{\partial \ell}{\partial F_K(x)}$ F_x

$L = y_i \log(P_i) + (1-y_i) \cdot \log(1-P_i)$ → binary log loss

$\boxed{P_i = \frac{1}{1+e^{-\hat{y}_i}}}$ → logistic function

$\boxed{\frac{\partial L}{\partial \hat{y}_i} = \frac{y_i - (1-y_i) \cdot e^{\hat{y}_i}}{1+e^{\hat{y}_i}}}$

After multiplying by $\frac{e^{-\hat{y}_i}}{e^{-\hat{y}_i}}$

$$L' = \frac{y_i \cdot e^{-\hat{y}_i} + y_i - 1}{1 + e^{-\hat{y}_i}} = \frac{y_i \cdot (1 + e^{-\hat{y}_i})}{(1 + e^{-\hat{y}_i})} - \frac{1}{1 + e^{-\hat{y}_i}}$$

$$= y_i - P_i$$

$$\boxed{-\frac{\partial L'}{\partial f_k(i)} = P_i - y_i}$$

P_i = probability of $x_i \in$ class 1

$y_i = 0$ or 1

$P(x_i \in 1)$

pseudo residual $= (P_i - y_i) \approx$ error/residual.

In case of SQ loss $\leftarrow \underbrace{y_i}_{\text{(actual value)}} - \underbrace{F_k(x_i)}_{\text{(estimated value)}}$

$y_i - \hat{y}_i$

~~In case of log loss~~

Q pseudo-residual $\approx$ residual/error always

For SQ ~~loss~~ $\rightarrow$ Yes

For log-loss $\rightarrow$ Yes

$\boxed{\text{any - loss} \rightarrow \text{No.}}$

It's not always true that pseudo-residual can be interpreted as error/residual.

Q why are we using pseudo-residual if (any loss → No.) from previous question but can't we use residual only?

Sol → Proof ÷

$D\{x_i, y_i\}$

$\mathcal{L}$

F

$F(x_i) = y_i$

minimizes loss fn

In boosting we build a sequence of models

$$F_0(x) = \arg\min \sum_{i=1}^{n} L(y_i, \gamma)$$

$\mathcal{L} \subset$ SQ-loss

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

In case regression γ would be average of y_i

H = set of all base learners

$$F_m(x) = F_{m-1}(x) + \arg\min_{h_m \in H} \sum_{i=1}^{n} L\Big(y_i, \overbrace{\underbrace{F_{m-1}(x_i)}_{\text{old}} + h_m(x_i)}^{\hat{y}_i}\Big)$$

$\boxed{\text{Loss between } \mathcal{L}(y_i, \hat{y}_i)}$

$$\boxed{W_{new} = W_{old} - \eta \left(\frac{\partial L}{\partial W}\right)_{old}}$$

variable

functions

learning rate

gradient

update step Gradient descent

<u>Steepest descent</u>

$$W_{new} = W_{old} - \text{learnrate} \left(\frac{\partial L}{\partial w}\right)$$

$$F_m(x) = F_{m-1}(x) - \gamma_m \sum_{i=1}^{n} \nabla F_{m-1} L(y_i, F_{m-1}(x_i))_i$$

$$\left(\frac{\partial L}{\partial F_k(x)}\right)_{F_{m-1}(x)}$$

$$-\nabla F_{m-1}$$

$$\|$$

$\boxed{\text{pseudo residual}}$

<u>Gradient Boosting</u>

$\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss fn $L(y, F(x))$, number of iterations M

1.) Initialize model with a constant value

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma)$$

2.) Find $m = 1$ 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 (eg tree) $h_m(x)$ to pseudo-residuals,
i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$

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

4) Update the model:

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

3.) Output $F_m(x)$.

In GradientBoost Algorithm → we need high bias and low variance models that means in Decision Tree we need shallow decision tree as base learners.

Decision tree of depth 1 is called decision stump.

Regularization By Shrinkage

$$F_m(x) = h_0(x) + \sum \gamma_m h_m(x)$$

M: # of base models

base-models ↑ ⇒ overfitting ↑ → Var ↑

M ↑ → bias ↓

Incase of Gradient Boosting → Concept is called Shrinkage:

base model is hyper-parameter

CV.

$$\widetilde{F_m(x)} = \widetilde{F_{m-1}(x)} + \underbrace{\nu}_{\text{learning rate}} \gamma_m \underbrace{h_m(x)}, \quad 0 \le \nu \le 1$$

$v \approx 0.1$

↳ hyperplane → CV

If v is small less weightage is given to the model

$v \uparrow$ chances of overfitting and variance increases

2 hyperparameter ⌈ M # base models
⌊ v → shrinkage coefficient

CV → gridsearch (M, v) → It will reduce the overfitting

→ best value of M, v should be selected using grid search

shrinkage

Base models.

<u>Train & Run time complexity</u> GB

Train : $O(n \lg n \, d * M)$ M : # base learners.

RF :- D — D_1 — M_1
— D_2 — M_2
⋮
D_k — M_k

} Trivially parallelizable

GBDT :- not easy to parallelize → because it is a serial algorithm (step by step process).

Stage 0 → M_0
↓
Data 1
↓
Stage 1 → M_1
↓
Data 2
↓

In general GBDT take more time to train than Random forest.

Runtime:

GBDT: $O(\overbrace{\underbrace{\text{depth}}_{\uparrow \text{ Small in } \underline{GB}} * M}^{\to \text{low latency application}})$ $\qquad O(\text{depth} * M + M) \simeq O(\text{depth} + M)$

$h_m(x)$

Space: $O(\overbrace{\text{store each tree}}^{\to \text{if..else}} + \underbrace{\gamma_m}_{\downarrow M})$

Suppose GBDT depth = 3
M = 200 → easy → just if else

$\boxed{\text{GBDT : is used by lot of Internet Companies}}$

⇨ $\boxed{\text{XG Boost : GBDT + row sampling + Column Sampling}}$

~~Storage data~~

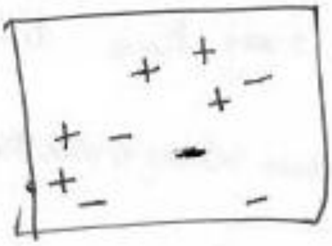
AdaBoost: Geometric Intuition

Used in computer vision or Image processing / face detection

Adaptive Boosting → i) Build learners sequentially
ii) samples misclassified by the previous learner weighted more in subsequent learners.

NOTE
Decision Stump is either parallel to x-axis or y-axis

D_1



D.T with depth = 1

$\Downarrow$

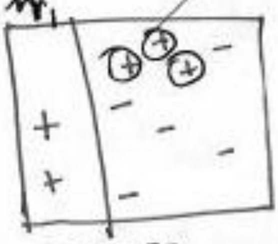
decision Stump

First round:

$\underline{\underline{\alpha_1 h_1}}$

These are errors

h_1



$\varepsilon_1 = 0.30$

weight $\rightarrow$ $\alpha_1 = 0.42$

D_2

weight on wrong point increased

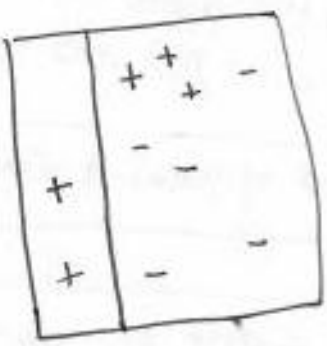
weighted models.

$\downarrow$

upsample these points

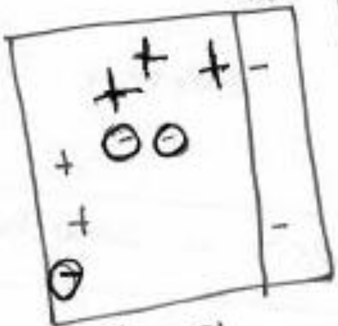
The second round

$h_1(x)$



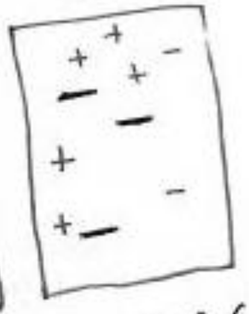
$\underline{\underline{\alpha_2 h_2}}$

h_2



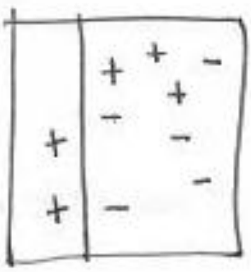
$\varepsilon_2 = 0.21$
$\alpha_2 = 0.65$

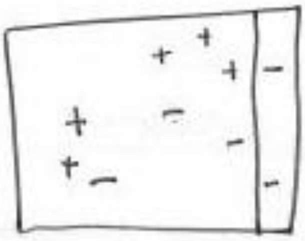
D_3

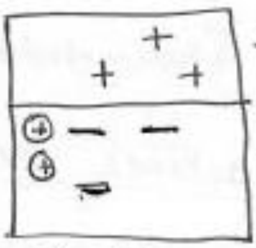


weight of (wrong) -ve classified points increased

The third round







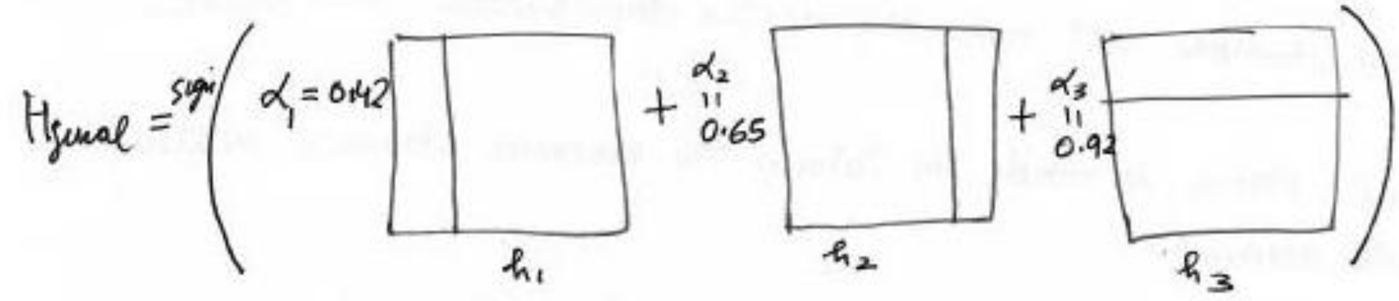
+ve

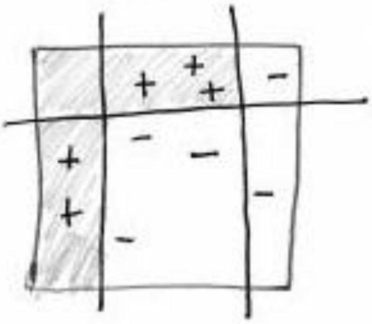
-ve

$\varepsilon_3 = 0.14$
$\alpha_3 = 0.92$

The final classifier

$F_3(x) = \alpha_1 h_1 + \alpha_2 h_2 + \alpha_3 h_3$





$\boxed{\text{In case of GBDT } \alpha \text{ are equivalent to } \gamma}$

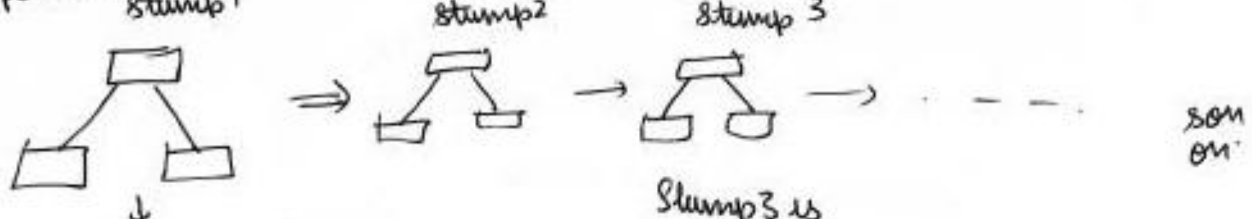
$\boxed{\text{In case of GBDT: we compute pseudo-residuals.}}$

AdaBoost → Forest of Trees made with Adaboost, the trees are usually just a node and two leaves

Tree with just one node and 2 leaves are called stumps.

Stumps are weak learners that means it has high bias.

Note → In a Forest of Stumps made with AdaBoost, order is important



Errors made by stump 1 it influences the stump2

Stump3 is influenced by stump 2

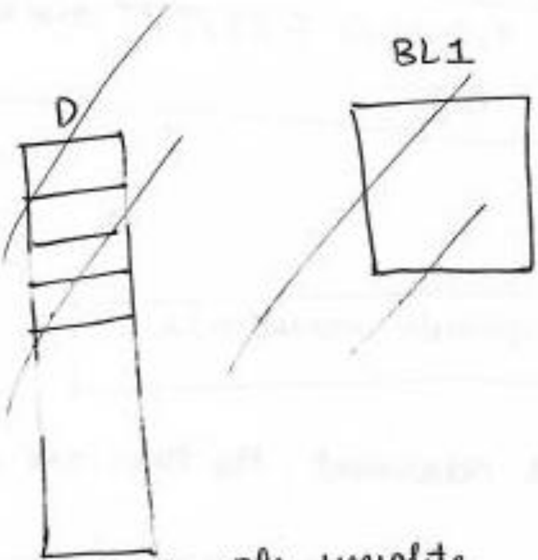
3 important things about AdaBoost

1.) It combines a lot of "weak learners" to make classifications. The weak learners are almost always stumps.

2.) Some stumps get more say in the classification than others.

3.) Each stump is made by taking the previous stump's mistakes into account.

Step 1



Step 1 → Assign sample weights.

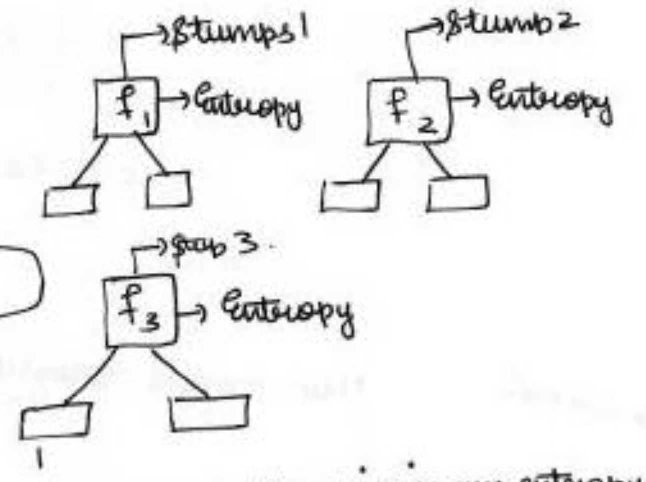
$w = 1/n = 1/7$

f_1	f_2	f_3	O/p	Sample weight
				1/7
				1/7
				1/7
				1/7
				1/7
				1/7
				1/7

Step 1 Sample weight = $1/n = 1/7$

f_1	f_2	f_3	O/P	Sample weight
			Yes	$1/7$
			No	$1/7$
			Yes	$1/7$
			No	$1/7$
				$1/7$
				$1/7$
				$1/7$

Incorrect



we select the minimum entropy / Gini among the 3 features

Step 2 → We calculate the total error of the selected stump
suppose correctly classified = 4
incorrect = 1

Step 3 → We calculate total error = $\frac{1}{7}$

$$\text{Performance of Stump} = \frac{1}{2} \log_e \left(\frac{1 - TE}{TE} \right)$$

$$= \frac{1}{2} \log_e \left[\frac{1 - 1/7}{1/7} \right]$$

$$= \frac{1}{2} \log_e [6]$$

$$= 0.896$$

We just need to increase weight of incorrectly classified stump on the other hand decrease weight of correctly classified stump using performance of stump

For incorrect:

$$\text{New sample weight} = \text{Weight} \times e^{\text{Performance say}}$$

$$= \frac{1}{7} \times e^{0.895}$$

$$= 0.349$$

For correct:

$$\text{New sample weight} = \text{Weight} \times e^{-\text{Performance say}}$$

$$= \frac{1}{7} \times e^{-0.895}$$

$$= 0.05$$

f_1	f_2	f_3	O/P	Sample Weight	Updated Weight
			Yes	$1/7$	0.05
			No	$1/7$	0.05
			Yes	$1/7$	0.349
			No	$1/7$	0.05
				$1/7$	0.05
				$1/7$	0.05
				$1/7$	0.05

For incorrect updated weight

Step-4

NOTE: Sum of updated weight is not equal to 1 so we need to normalize it.

f_1	f_2	f_3	O/P	Normalized weight
			Y	0.07
			N	0.07
			Y	0.51
			N	0.07
				0.07
				0.07
				0.07

Step 5 → We will create a new decision stump.

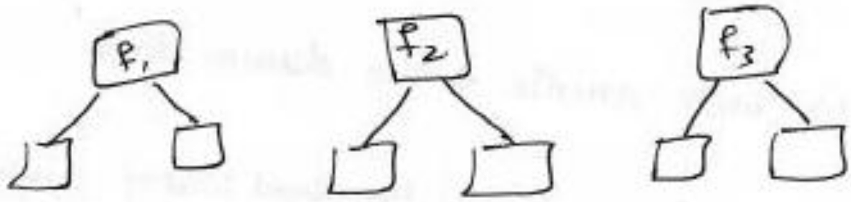
f_1	f_2	f_3	O/P	Normalized Weight	Cumulative freq
			Yes	0.07	0 - 0.07
			No	0.07	0.07 - 0.14
			Yes	0.51	0.14 - 0.65
			No	0.07	0.65 - 0.72
				0.07	0.72 - 0.79
				0.07	0.79 - 0.86
				0.07	0.87 - 0.93

We will use the sample weights to calculate weighted Gini Indexes to determine which variable should split the next stump.

f_1 f_2 f_3 O/P Normal Gini

if number (randomly selected) is between 0.07 – 0.14 we put this sample into new collection of sample
if number is between 0.08 – 0.93 we put corresponding sample into new collection of sample

f_1	f_2	f_3	O/P	Cumulative freq	Randomly select
			No	0.07–0.14	0.09
			Yes	0.85–0.93	0.84.



Same process is repeated

Gradient Boost Part 1:

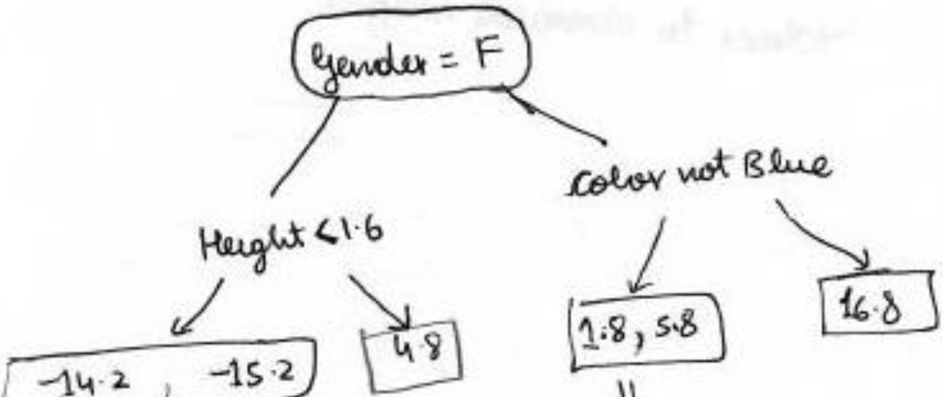
Height	Favourite Color	Gender	Weight
1.6	B	M	88
1.6	G	F	76
1.5	B	F	56
1.8	R	M	73
1.5	G	M	77
1.4	B	F	57

Step I. First Average Weight = 71.2

Step 2 → build a tree using error from previous tree

Residual = (Observed − 71.2)

Height	FC	Gender	Weight	Residual
1.6	B	M	88	16.8
1.6	G	F	76	4.8
1.5	B	F	56	−15.2
1.8	R	M	73	1.8
1.5	G	M	77	5.8
1.4	B	F	57	−14.2



$\frac{-14.2 - 15.2}{2} = -14.7$

$\frac{1.8 + 5.8}{2} = 3.8$

so the Predicted weight = 71.2 + 16.8 = 88 → It leads to low bias but very high variance

~~Height~~ Gradient Boost deals with this problem by using a learning Rate to scale the contribution from the new tree.

The learning Rate is a value between 0 and 1.

Now the predicted weight = 71.2 + (0.1 × 16.8) = 72.9

$\downarrow$ w

Residual = (Weight − (w_0 × ~~Average~~ Residual + Average))

H	F.C	G	Weight	Residual 1	Residual 2	Residual
			88	16.8	15.1	13.6
				4.8	4.3	3.9
				−15.2	−13.7	−12.4
				1.8	1.4	1.1
				5.8	5.4	5.1
				−14.2	−12.7	−11.4

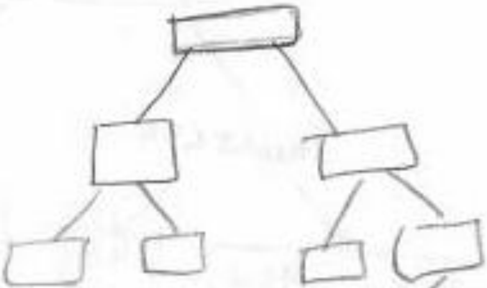
$\longrightarrow$ Residual getting smaller

For first data pt → $71.2 + (0.1 \times 16.8) + (0.1 \times 15.1)$

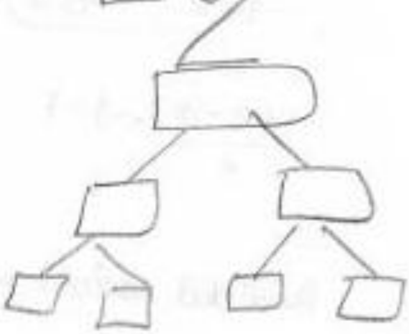
$= 74.4$

↳ closer to observed weight

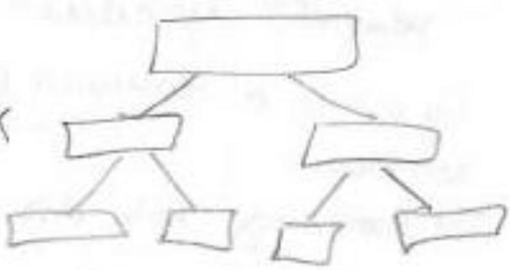
71.2 + 0.1 ×



+ 0.1 ×



+ 0.1 ×



Regression details for Gradient boost.

Height	F.C	Gender	Weight(kg)
1.6	Blue	M	88
1.6	Green	F	76
1.5	Blue	F	56

← x_i (Height, F.C, Gender)
— y, weight

Data $\{(x_i, y_i)\}_{i=1}^{n}$ and a differentiable loss fn $L(y_i, F(x))$

loss function for Gradient boost $= \frac{1}{2}(\text{Observed} - \text{Predicted})^2$

$$\frac{d}{d\,\text{Predicted}} = \frac{2}{2}(\text{Observed} - \text{Predicted}) \times -1$$

$$= -(\text{Observed} - \text{Predicted}).$$

Step 1 → $\underline{F_0}(x) = \underset{\gamma}{\text{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

↓ loss fn ↘ predicted value

$\frac{1}{2}$-(88 − Predicted) + −(76-Predicted) + (56-Predicted) = 0

3Predicted = 88+76+56

$F_0(x)$ Predicted $= \frac{88+76+56}{3}$

$F_0(x) = 73.3$

leaf is predicted

Step 2: for $m = 1$ to M: Calculate residual

A.) Compute $r_{im} = -\left[\dfrac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x) = F_{m-1}(x)}$

$= (\text{Observed} - F_0(x))$

$r_{1,1} = (\text{Observed} - 73.3) = (88 - 73.3) = 14.7$

$= (76 - 73.3) = 2.7$

$= (56 - 73.3) = -17.3$

Thus $r_{i,m}$ are called pseudo residuals

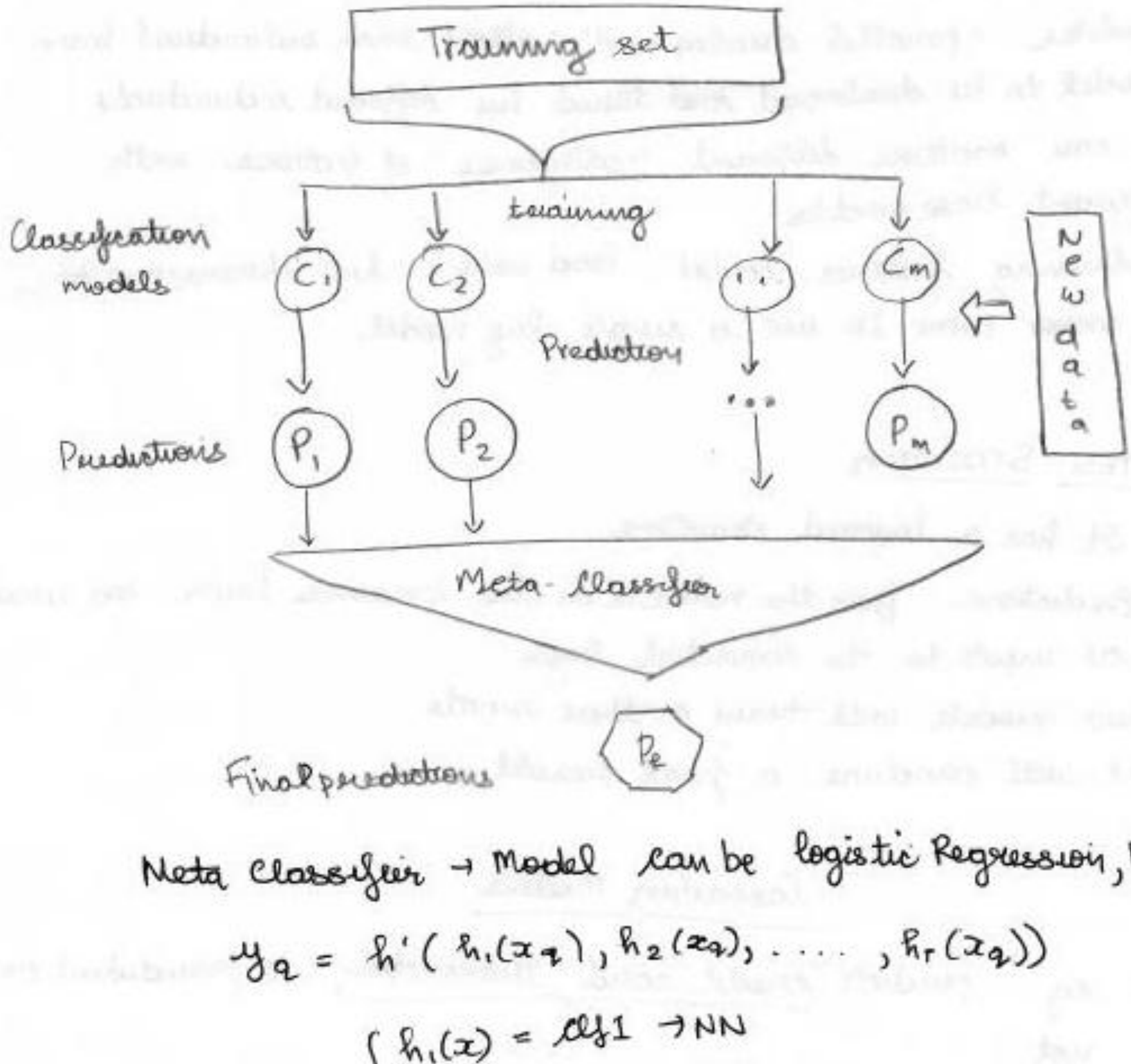
B.) Fit a regression tree to the r_{im} values and create terminal regions R_{jm} for $j = 1 \ldots J_m$.

C.) For $j = 1 \ldots J_m$ compute $\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$

Here we are taking previous prediction.

Stacking Classifier

Overview



Meta Classifier → Model can be logistic Regression, Decision Tree

$$y_q = h'(h_1(x_q), h_2(x_q), \ldots, h_r(x_q))$$

meta learner
$$\begin{cases} h_1(x) = \text{Clf}1 \rightarrow \text{NN} \\ h_2(x) = \text{Clf}2 \rightarrow \text{RF} \\ h_3(x) = \text{Clf}3 \rightarrow \text{GNB} \end{cases}$$

meta learner ← $h'(x) = \cancel{\text{RF}}$ Logistic Regress

Benefits of Stacking

→ Stacking increases the diversity of the algorithm and models used

→ Stacking can decrease bias - rather than winner takes all, combine datasets to decrease bias.

→ Enables "parallel development": allow each individual base model to be developed and tuned by different individuals.

→ We can capture different "categories" of features with different base models.

→ Combining features could lead very high dimensionality, if ~~were~~ were to use a single big model.

NOTES STACKING

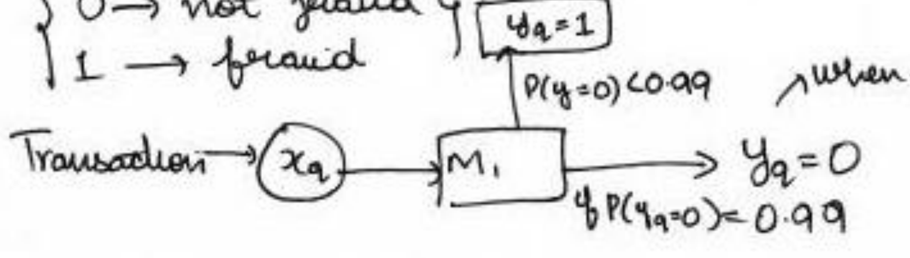
- It has a layered structure
- Predictions from the models in the previous layer are used as inputs to the sequential layer.
- new models will train on these inputs
- It will produce a final result

Cascading Models:

→ eg: predicts credit card transaction is fraudulent or not.

Transaction → x_q (vector) {consisting → location, creditscore}
Credit card company have lot of domain knowledge

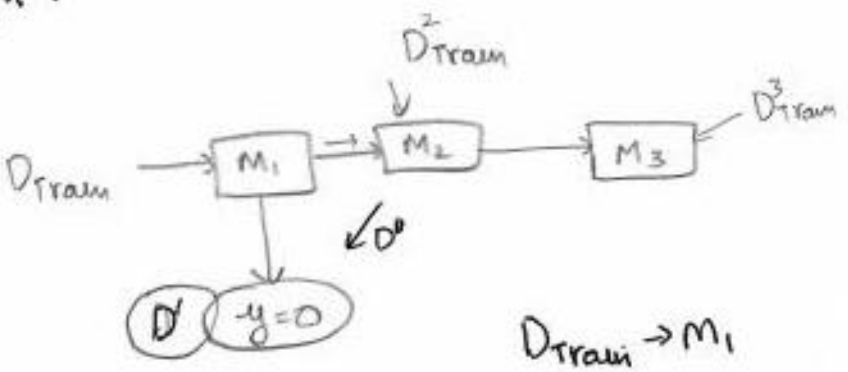
{0 → not fraud, 1 → fraud}



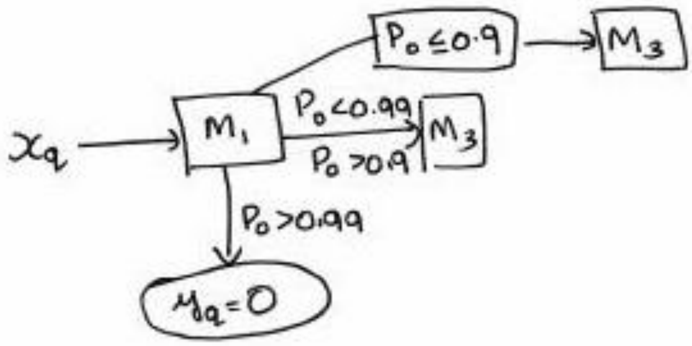
when probability of $y_q = 0$ is 0.99 than not fraud

Cascading model:

Typically used when the cost of making a mistake is high.



$$D_{Train} \rightarrow M_1$$
$$D^2_{Train} \rightarrow M_2$$
$$D^2_{Train} = D_{Train} - D'$$
$$D^3_{Train} = D^2_{Train} - D''$$



if $P_0 > 0.99$
    $y_q = 0$
else { if $P_0 > 0.9$
        use M_2
    else
        use M_3.
}

$P_0 = P(y_q = 0)$

$P_1 = P(y_q = 1)$

Cascade being built for fraud detection.