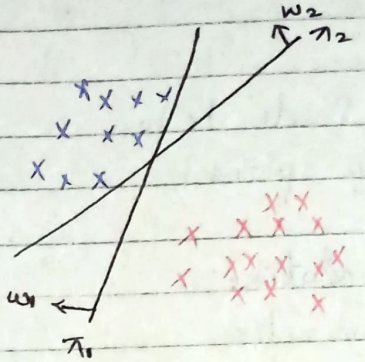
Module-4

CH- SVM

→ Geometric intution :-

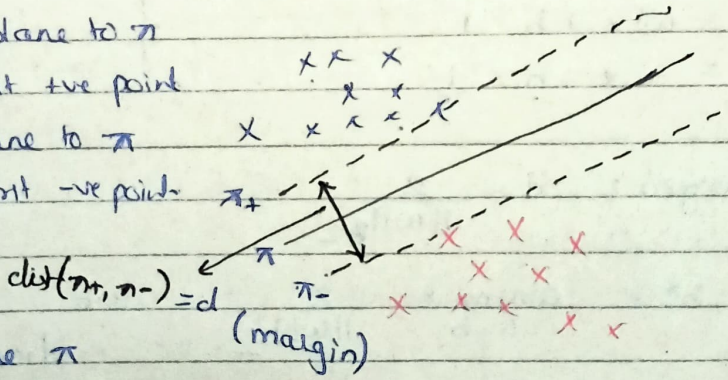
SVM is a popular ML algorithm, it can do both classification and regressions.



There are many hyperplane that separate +ve & -ve point.

So which of one is to be preferred.

→ The key idea of SVM:- try to find the hyperplane that separate +ve & -ve points as wide as possible.: in above plot π_2 is more ~~prefe~~ preferable because π_1 is very close to points. such a plane which separate the points with maximum margin is called margin maximizing hyperplane.



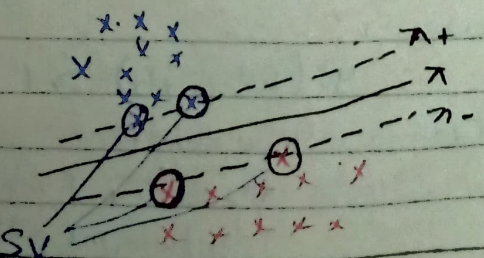
π_+ is a parallel plane to π which touches the first +ve point

π_- is a parallel plane to π which touches the first -ve point.

We want a hyperplane π s.t dist(π_+, π_-) is maximum.

SVM:- Try to find the hyperplane that maximizes the dist(π_+, π_-) because as margin ↑, there is less chance of misclassificatio

SV(support vectors) :- The points through which π_+ & π_- passes are called support vector



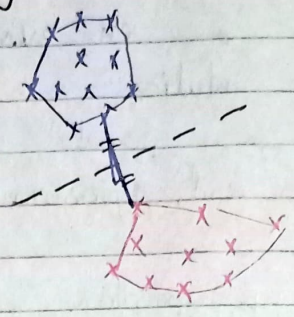
→ Alternative geometric interpretation of SVM:-

① Build a convex hull for the +ve point
-ve

② Find the ~~line~~ shortest line connecting these hulls

③ Bisect the line

④ The plane that bisects the line is margin maximizing plane.

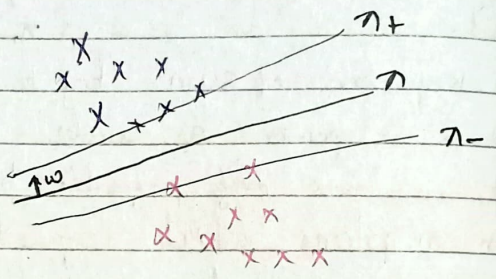


☆ Mathematical foundation:-

π :- margin - maximization

π : $w^T x + b = 0$

As π^+ & π^- are ~~normal to~~ || to π the w will also be normal to π^+ & π^-



π : $w^T x + b = 0$

If π^+ : $w^T x + b = 1$

π^- : $w^T x + b = -1$

} w need not be unit vector.

Margin : $d = \frac{2}{\|w\|_2}$

$w^*, b^* = \underset{w,b}{\text{argmax}} \frac{2}{\|w\|_2}$ s.t : all the positive points lies ~~under~~ above π^+ and all negative points are below π^-

↳ Let x_i be a ~~point~~ +ve point

Then distance of this point from π will be ≥ 1 because w^T ~~b~~ π^+ : $w^T x + b$ and π^+ is boundary of +ve point here any point in the region will have

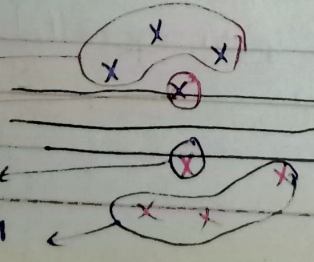
$y_i(w^T x_i + b) \geq 1$

[x :- $y_i(w^T x_i + b) > 1$ <

$y_i(w^T x_i + b) = 1$ <

$y_i(w^T x_i + b) = 1$ <

$y_i(w^T x_i + b) > 1$ <



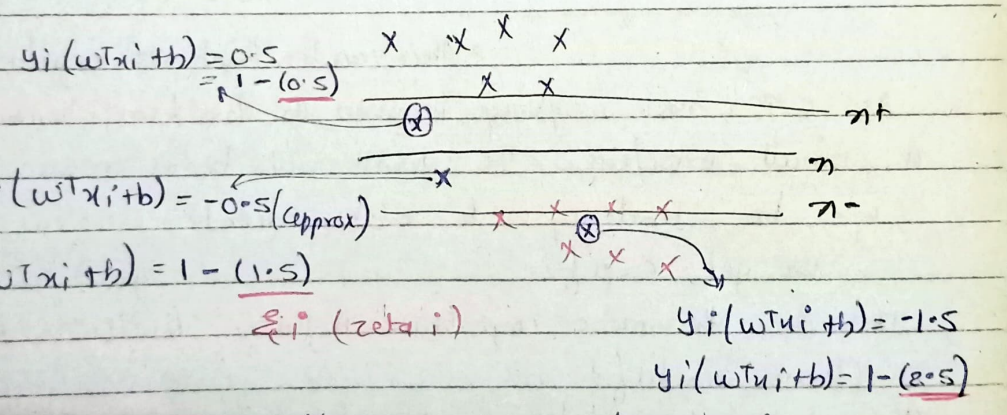
Hence for SV: $y_i(w^T x_i + b) = 1$, for all other points: $y_i(w^T x_i + b) > 1$

$$w^*, b^* = \underset{w,b}{\text{argmax}} \frac{2}{\|w\|_2} \quad s.t \quad y_i(w^T x_i + b) \geq 1 \quad \forall x_i$$

So, we have constraint optimization problem.
But there is one problem. we have considered only linearly separable data. what if our data is non-linearly separable or almost linearly separable in that case our optimization eq will fail
Hence the above equation is called Hard margin SVM

$$\boxed{w^*, b^* = \underset{w,b}{\text{argmax}} \frac{2}{\|w\|} \quad s.t \quad y_i(w^T x_i + b) \geq 1 \quad \forall x_i}$$

Hard maigin SVM.
So, can we change our equation slightly to, so that our mode also works for almost linearly separable data.



So, we create a new variable ξ_i $\forall$ the points s.t if
- +ve part lies in +ve region $\xi_i = 0$
- -ve " " " -ve " $\xi_i = 0$

~~for point this~~ anywhere

ξ_i for all other point is +ve.
As $\xi_i \uparrow$ the point is feather away from its correct hyperplane in incorrect direction.

$\forall x_i \longrightarrow \xi_i = 0$ if $y_i(w^T x_i + b) \geq 1$

$\xi_i > 0$ if is equal to some threshold and distance away from correct hyperplane in incorrect direction.

we also know that $\text{argmax } f(x) = \text{argmin } \frac{1}{f(x)}$

$$w^*, b^* = \underset{w,b}{\text{argmin}} \frac{\|w\|_2}{2} + C \cdot \frac{1}{n} \sum_{i=1}^{n} \xi_i$$

$$\text{s.t } y_i (w^T x_i + b) \geq 1 - \xi_i \;\; \forall i \text{ and } \xi_i \geq 0$$

we want to minimize error or misclassification and ξ_i is >0 for incorrectly classified point.

C is hyper parameter.

$$w^*, b^* = \underset{w,b}{\text{argmin}} \frac{\|w\|}{2} + C \left(\frac{1}{n} \sum_{i=1}^{n} \xi_i \right)$$

hyper parameter

distance of misclassified point

average distance of misclassified point

$\frac{1}{\text{margin}}$

$$\text{s.t } y_i (w^T x_i + b) \geq 1 - \xi_i \;\; \forall i \text{ and } \xi_i \geq 0$$

Loss: This has to be minimized. (Hinge loss)

This can be think as regularization

As $C \uparrow$ more weightage is given to loss term, hence for even a small error/loss the term will become more. Hence we have tendency to make mistake decrease hence we get overfit.

If $C \downarrow$ the more importance is given $\frac{\|w\|_2}{2}$, then we have underfit.

The above formulation of sum is called soft-margin SVM.

$$w^*, b^* = \underset{w,b}{\text{argmin}} \frac{\|w\|_2}{2} + C \cdot \frac{1}{n} \sum_{i=1}^{n} \xi_i \quad \text{s.t } y_i (w^T x_i + b) \geq 1 - \xi_i \;\&\; \xi_i \geq 0$$

soft-margin SVM.

* Why +1 and -1 for π^+, π^-

π : $w^T x_i + b = 0$ $\quad \pi^+ : w^T x_i + b = +1$ $\quad \pi^- : w^T x_i + b = -1$

why?

* w need not be unit vector.

$$\text{margin} = \frac{2}{\|w\|}$$

Lets assume instead of +1 we took K, K>0

$\pi^+ : w^T x_i + b = K$

$\pi^- : w^T x_i + b = -K$

margin :- $\frac{2K}{\|w\|}$

$\underset{w,b}{\text{argmax}} \frac{2}{\|w\|} = \underset{w,b}{\text{argmax}} \frac{2K}{\|w\|}$ because (2, 2K are constant here no effect)

∴ we can take any value of K, But for simplicity we take K=1

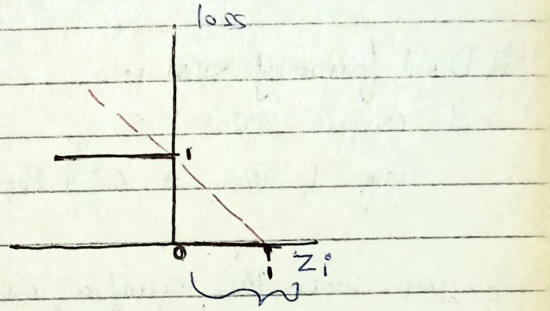
★ loss-minimization interpretation :-

While studying in LR, we learnt that As 0-1 loss is not continous we took an approximation and that approximation is logistic loss.

Logistic reg :- logistic loss + reg

Linear fn :- linear loss + reg

SVM :- hinge loss + reg



z_i in svm is $y_i(w^T x_i + b)$

if $y_i(w^T x_i + b) > 0$ that means point is correctly classified. hence loss = 0

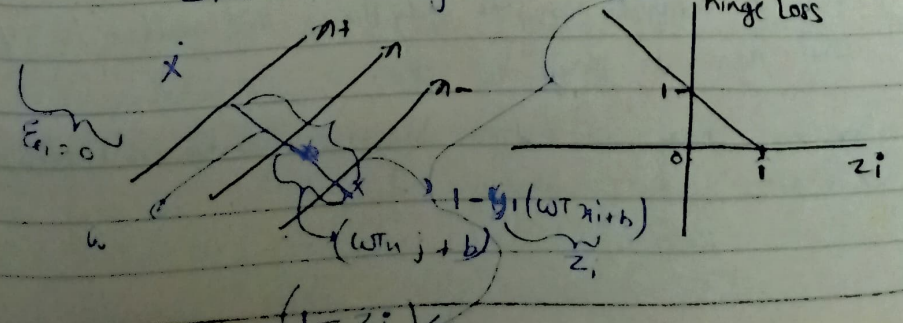
when $z_i < 0$:- x_i is incorrectly classified hence in 0-1 loss we give it a loss of 1.

And for SVM we use hinge loss as approximation

So, how does our hinge loss behave.

If $z_i \geq 1$, hinge loss = 0

$z_i < 1$, hing loss = $1 - z_i$



Hence hinge loss is same as ξ_i

Less min. SVM formulation is

$$\min_{w,b} \sum_{i=1}^{n} \max(0, 1 - \underbrace{y_i(w^T x_i + b)}_{z_i}) + \underbrace{\lambda \|w\|^2}_{\text{Regularization}}$$

$\lambda \uparrow$ underfit
$\lambda \downarrow$ overfit

is same as soft-margin SVM, behave opposite $C \uparrow$ overfit, $C \downarrow$ underfit.

$$\min_{w,b} \frac{\|w\|}{2} + C \sum_{i=1}^{n} \xi_i \quad \text{s.t} \quad (1 - y_i(w^T x_i + b)) \geq \xi_i, \; \xi_i \geq 0$$

★ Dual form of SVM :-

soft-margin SVM:

$$\min_{w,b} \frac{1}{2}\|w\| + C \sum_{i=1}^{n} \xi_i \quad \text{s.t} \quad y_i(w^T x_i + b) \geq 1 - \xi_i, \; \xi_i \geq 0$$

we can write this equation as

$$\max_{\alpha_i} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j \quad \text{s.t} \quad \alpha_i \geq 0, \; \sum_{i=1}^{n} \alpha_i y_i = 0$$

solving both the equation is equivalent.

→ This formulation is called Dual form.

(i) $\forall x_i$ we have α_i

(ii) x_i only occur in form of $x_i^T x_j$

(iii) $\hat{y}_q = f(x_q) = \sum_{i=1}^{n} \alpha_i y_i x_i^T x_q + b$

(iv) $\alpha_i > 0$ only for SV, $\alpha_i = 0$ for non support vectors.

$$f(x_q) = \sum_{i=1}^{n} \alpha_i y_i x_i^T x_q + b$$

↳ if $\alpha = 0$ whole thing is zero.

Here to compute x_q the only thing that matter is SV and non-SV plays no role.

Thats why the name is SVM because only SV play role in classification.

$$\max_{\alpha_i} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

s.t $\alpha_i \geq 0$ → $\alpha_i = 0$ for non-SV, $\alpha_i > 0$ for SV

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

$x_i^T x_j = x_i \cdot x_j$ = cosine-similarity b/w x_i & x_j {if $\|x_i\| = 1, \|x_j\| = 1$}

we can replace $x_i^T x_j$ with any similarity not just cosine-similarity. That makes the SVM super popular because we can plug in any similarity measure in dual form equation.

often time we use $K(x_i, x_j)$ it is called Kernel fun.

$$\max_{\alpha_i} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

→ Kernel funct.

$$f(x_q) = \sum_{i=1}^{n} \alpha_i y_i K(x_i, x_q) + b$$

☆ Kernel trick —

Dual form:-

$$\max_{\alpha_i} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j \quad \text{s.t} \sum_{i=1}^{n} \alpha_i y_i = 0 \; ; \; \alpha_i \geq 0$$

→ can be replaced by any similarity funct. One popular class of funct. is Kernel.

replacing $x_i^T x_j$ with $K(x_i, x_j)$ is called Kernel trick.

→ The most important idea in SVM is Kernel-trick

→ if we not replace $x_i^T x_j$ with any other Kernel then it is called Linear SVM. otherwise it is called Kernel SVM.

⇒ Q#

Linear-SVM:- find margin-max hyperplane in x_i's space

Logis-reg:- min logistic loss in x_i's space:.

Both of them approx give similar results But the world changing idea in SVM became the Kernel trick.

Generalization —

In linear SVM we try to separate with hyperplane But if data is not linearly separable, Linear SVM will fail In Logistic regression also we need feature transform to succeed But Kernel SVM will succeed if we identify the correct Kernel.

Kernel trick do a similar task as that of feature engineering in logistic regression.

Hence Kernel-SVM : can separate non-linear data also.

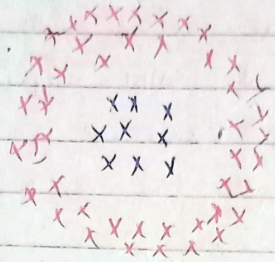
Note : there is a diff b/w feature engineering & Kernel trick

☆ Polynomial Kernel :-

In Logistic Reg we had seen that we can apply feature transformation

$f_1' = f_1^2$

$f_2' = f_2^2$



to make the data linearly separable

Now, let's look how Kernalisation works on this.

$$K(x_1, x_2) = (x_1^T x_2 + c)^d$$

Ex:- Quadratic Kernel

$$K(x_1, x_2) = (1 + x_1^T x_2)^2$$

let $x_1 : \langle x_{11}, x_{12} \rangle$

$x_2 : \langle x_{21}, x_{22} \rangle$

$$= (1 + x_{11}x_{21} + x_{12}x_{22})^2$$

$$= 1 + x_{11}^2 x_{21}^2 + x_{12}^2 x_{22}^2 + 2x_{11}x_{21} + 2x_{12}x_{22} + 2x_{11}x_{21}x_{12}x_{22}$$

$$\hookrightarrow [1 + x_{11}^2 + x_{12}^2, \sqrt{2}x_{11}, \sqrt{2}x_{12}, \sqrt{2}x_{11}x_{12}] \rightarrow x_1'$$

$$\hookrightarrow [1 + x_{21}^2 + x_{22}^2, \sqrt{2}x_{21}, \sqrt{2}x_{22}, \sqrt{2}x_{21}x_{22}] \rightarrow x_2'$$

$$\rightarrow (x_1')^T (x_2')$$

$x_1 \rightarrow x_1'$; $x_2 \rightarrow x_2'$; $x_1^T x_2 \rightarrow x_1'^T x_2'^T$

Kernalization : to a internal feature transformation.

we don't do external feature transformation.

Kernal trick :- take a d dim data and change it to d' data internally using feature transformation

$d \rightarrow d'$

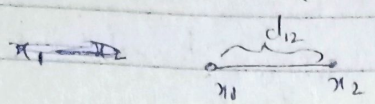
$d' > d$

But we need mgU Kernel.

* Radial Basis Function (RBF) Kernel:-

There are the most imp & general Kernel in SVM:-

$$K_{RBF}(x_1, x_2) = \exp\left(\frac{-\|x_1 - x_2\|^2}{2\sigma^2}\right)$$ → hyperparameter..

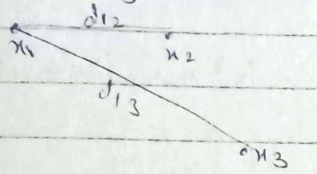


$\|x_1 - x_2\|^2$ = square of distance b/w x_1 & x_2

σ = hyper parameter.

- as distance increase $K_{RBF}(x_1, x_2)$ ↓ & $K(x_1, x_2) = \frac{1}{e^{\|x_1 - x_2\|^2 / 2\sigma^2}}$

This is behaving like some sort of similarity.

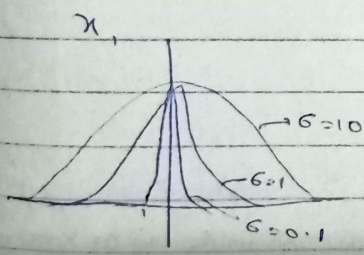


$d_{12} < d_{13}$

$K(x_1, x_2) > K(x_1, x_3)$

That mean if point are very far, $K(x_i, x_i)$ approaches 0 i.e. similarity becomes 0

- If σ increase, more points here are similar to here if σ is very large then all the points will be similar and then underfit.



If σ is very less then all the points will be non-similar then overfit.

σ can be treated as K in the KNN.

K-NN is similar to RBF Kernel SVM.

Hence RBF is said general Kernel SVM because it is very much similar to KNN.

If we remember, time complexity was only drawback of KNN, the RBF Kernel do a similar task with very less time complexity.

∴ RBF SVM are nice approximation to KNN.

⇒ If you don't know best Kernel to use, just simply use RBF.

But remember, we have a 2 hyperparameters if we use RBF, i.e. γ & C
↳ from soft margin SVM equation

★ Domain specific Kernels :-
There are lot of Domain specific Kernels:- like
String kernel for text classification
Genome kernel for Genome prediction problems.
Graph Kernel for Graph based problems.

★ Time complexity :-
Train :- ⟶ using SGD
↳ specialized algo like (sequential minimal optim) (smo)

$\approx O(n^2)$ for Kernel SVMs
when n is very large $O(n^2)$ is very time consuming

Test/Runtime :- $f(x_q) = \sum_{i \in I} \alpha_i y_i K(x_i, x_q) + b$
↳ 0 for non-SV

$O(k d)$
↳ # of SV ↳ dimensionality.
$1 \le k \le n$

★ nu-SVM :- Standard SVM formulation with C as hyper parameter
Here our only said $C \ge 0$ nothing else. Here is alternative formulation of SVM, called nu-SVM
nu lies b/w 0 & 1 $\quad 0 \le nu \le 1$
★ nu is an upperbound of fraction of error
nu is an lower bound of support vectors.
Ex:- we don't want >10% error. we simply set nu = 0.1

nu max fraction of error that can happen.
If nu = 0.01 ⇒ % of error $\le$ 1%.
↳ # of SV $\ge$ 1% of n data points.

Support Vector Regression (SVR)

We can do SVR with slight changes.

$$\min_{w,b} \frac{1}{2}\|w\|^2 \quad \text{s.t} \quad y_i - (w^T x_i + b) \leq \epsilon$$
$$(w^T x_i + b) - y_i \leq \epsilon$$
$$\epsilon \geq 0$$

This is the linear formulation of SVR, we can also have kernel formulation

ϵ is hyper parameter.

If $\epsilon \downarrow$ then errors are low on training data ∴ overfitting ↑

$\epsilon \uparrow$ underfitting may increase.

Cases :-

1. Feature engg & FT :- Kernel design.
2. Decision surface :- Hyperplane for linear SVM, non-linear surface using Kernel.
3. Similarity & Distance fn :- Convert to kernel & then plug it in SVM.
4. Interpretability & Feature imp :- No implicit way, hence forward feature selection is necessary.
   → Kernel case.
   In linear SVM it is easy to get feature imp.
5. Outlier :- very little impact
   But if we are using RBF with small σ then it may impact.
6. Bias-variance tradeoff :- As $C \uparrow$ overfit, $C \downarrow$ underfit.
7. Large d :- its good for SVM :- They works very well in higher dimension :-

→ Best case :- If we have a right Kernel

→ Worst case :- ① n is large → train time is high,
② K is large (K = no. of SV)

CH: Decision Trees:-

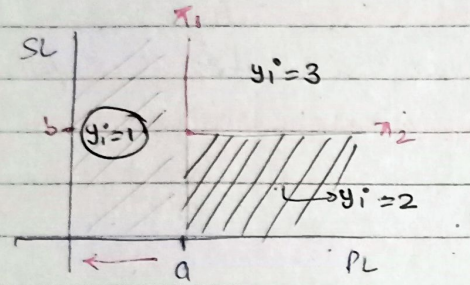
★ Geometric Intuition:- Since we have learned algorithms like KNN, NB, Log reg, LR, reg, SVM

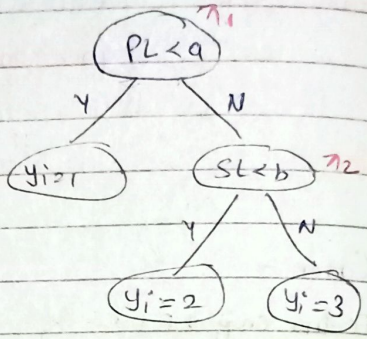
(instance based method) (probabilistic method) (geometric, because they try to find hyperplane)

Decision Tree is some what like If else condition.

DT is nested If.. else classifier.

Ex Example DT of iris dataset





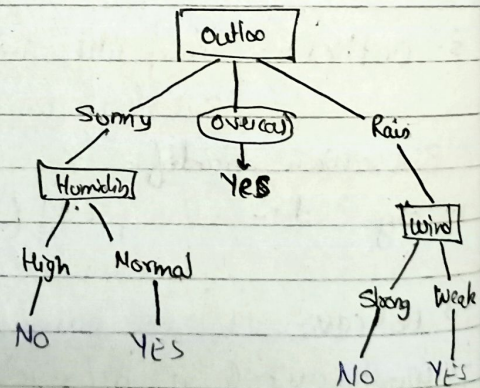
- all the hyperplanes are axis parallel.

∴ DT is set of axis parallel hyperplane.

Example:-

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
S	H	H	True	No
Overcast	H	H	False	Yes
Rainy	Mild	H	F	Yes
R	Cool	Normal	F	Yes
R	C	N	T	No
O	C	N	T	Yes
S	M	H	F	No
S	C	N	F	Yes
R	M	N	F	Yes
S	M	N	T	Yes
O	M	H	T	Yes
O	H	N	F	Yes
R	M	H	T	No

x_i $y_i \in \{0,1\}$



* Building a Decision Tree: In order to build a DT, we need to have understanding of some of the concepts

1. Entropy: Let we have random variable Y which can take K value $y_1, y_2 \dots y_K$ then entropy of Y is

$$H(Y) = -\sum_{i=1}^{K} P(y_i) \log_b(P(y_i)) \quad \begin{cases} b=2 \text{ or} \\ b=e=2.718 \end{cases}$$

$p(y_i) = P(Y=y_i)$

In our previous example: suppose $Y = \text{play} = \{\text{yes}, \text{No}\}$

$P(y=\text{yes}) = \frac{9}{14}$ $\quad P(y=\text{No}) = \frac{5}{14}$

$$H(Y) = -\frac{9}{14}\log\left(\frac{9}{14}\right) - \frac{5}{14}\log\left(\frac{5}{14}\right)$$

$\frac{9}{14}$ → % of +ve points; $\frac{5}{14}$ → % of −ve points

$\frac{\#\text{ of +ve Points}}{\text{Total \# point}}$ = % of +ve point

$H(Y) = 0.94$

properties of Entropy:- $Y \in \{0, 1\}$ (2 class R.V)

Case 1: If 99% of points are y_+ and only 1% of points are y_- in given dataset

$$H(Y) = -0.99\log(0.99) - 0.01\log(0.01) = 0.0801$$

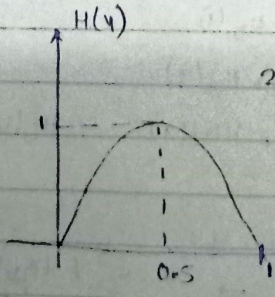
Case 2: $D \rightarrow y_+ \rightarrow 50\%$, $y_- \rightarrow 50\%$ $\quad H(Y) = -0.5\log 0.5 - 0.5\log 0.5 = 1$

Case 3: $D \rightarrow y_+ \rightarrow 0\%$, $y_- \rightarrow 100\%$ $\quad H(Y) = -0*\log() - 1*\log(1) = 0$

Now, let's look at ~~maximum~~ multiclass case

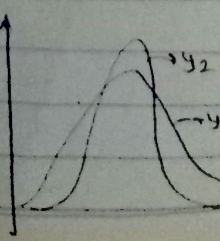
$Y \rightarrow y_1, y_2 \dots y_K$

if all these equally probable then entropy is maximum



2-class case.

By seeing the PDF of a random variable we can guess the entropy



$H(Y_2) < H(Y_1)$

more the peaked PDF less will be the entropy

(2) Information gain :- Now look back to our Example data set.

Outlook	Temp	Hum	Win	play
S	H	H	F	No
S	H	H	T	No
S	M	H	F	No
S	C	N	F	Yes
S	C	N	T	Yes

$E \Rightarrow -\frac{2}{5}\log\left(\frac{2}{5}\right) - \frac{3}{5}\log\left(\frac{3}{5}\right)$

$= 0.97$

Outlook → Sunny, Overcast, Rainy

O	H	H	F	Yes
O	C	N	T	Yes
O	M	H	T	Yes
O	H	N	F	Yes

$E = 0$

R	M	H	F	Yes
R	C	N	F	Yes
R	C	N	T	No
R	M	N	F	Yes
R	M	H	T	No

$E = 0.97$

$$\text{Average} = \frac{0.97 + 0 + 0.97}{3} = 0.64$$

Information gain $IG(Y, \text{outlook}) = \left(\frac{5}{14} \times 0.97 + \frac{4}{14} \times 0 + \frac{5}{14} \times 0.97\right) - 0.94$

$\frac{5}{14}$: No. of points in that division / Total points

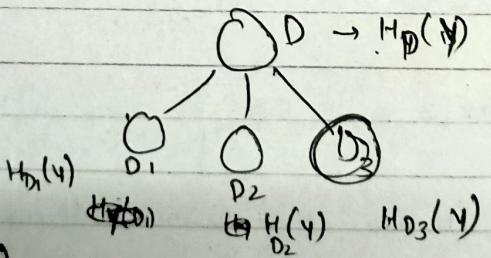
0.97 : Entropy of that division

$\frac{5}{14} \times 0.97 + \frac{4}{14} \times 0 + \frac{5}{14} \times 0.97$: weighted entropy

0.94 : Entropy of whole Data set.

$IG(Y, \text{outlook}) = \frac{5}{14} \times 0.97 - 0.94$

$IG(Y, D) = \frac{D_1}{D} * H_{D_1}(Y) + \frac{D_2}{D} * H_{D_2}(Y) + \frac{D_3}{D} * H_{D_3}(Y) - H_D(Y)$



$Y \rightarrow y_1, y_2 \ldots y_k$

$IG(Y, \text{val}) \Rightarrow \sum_{i=1}^{k} \frac{|D_i|}{|D|} H_{D_i}(Y) - H_D(Y)$

(2) Gini Impurity :- It is very similar to entropy

$$I_G(Y) = 1 - \sum_{i=1}^{n} (P(y_i))^2$$

Ex:- $Y \in \{y^+, y^-\}$

Case 1:- $P(y+) = 0.5 \qquad P(y-) = 0.5$

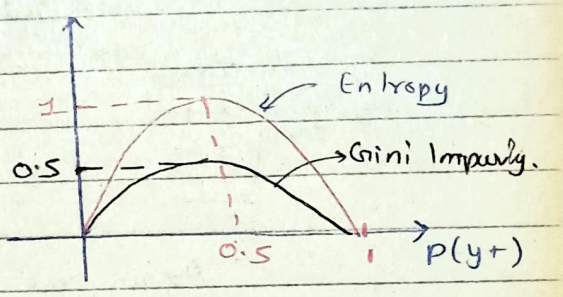
$$I_G(Y) = 1 - (0.5)^2 + (0.5)^2 \Rightarrow 0.5$$

$$H(Y) = 1$$

Case 2:- $P(y^+) = 1 \qquad P(y^-) = 0$

$$I_G(Y) = 1 - (1+0) = 0$$

$$H(Y) = 0$$



As we can see that both entropy and Gini impurity are very much similar only diff is max value of entropy is 1 and gini value = 0.5

So, why we use Gini Impurity.

$I_G(Y)$	$H(Y)$
$1 - \{P(y+)^2 + P(y-)^2\}$	$-P(y+)\log P(y^+) - P(y^-)\log(P(y^-))$

Entropy need log calculation which is max time taking than square calculation. Hence $I_G(Y)$ is max computationally efficient than entropy.

→ Construct a DT :- In our dataset of play or not we have

$$D \begin{cases} \text{Yes} \rightarrow 9 \\ \text{No} \rightarrow 5 \end{cases} \quad \therefore H_D(Y) = 0.94$$

Now, we have 4 features to choose using which we want to break the Data :-

① Outlook
- Sunny: Yes 2, No 3 → $H_{D_1}(Y) = 0.97$
- Overcast: Yes 4, No 0 → $H_{D_2}(Y) = 0$
- Rainy: Yes 3, No 2 → $H_{D_3}(Y) = 0.97$

IG ⇒ weighted entropy − $H_D(Y)$

= [illegible] 0.25

② Temperature
- hot: Yes: 2, No: 2 → $H_{D_4}(Y) = 0.1$
- mild: Yes: 4, No: 2 → $H_{D_5}(Y) =$
- cool: Yes: 3, No: 1 → $H_{D_6}(Y) =$

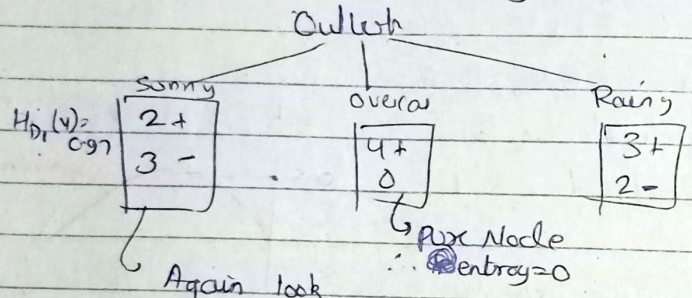
$I_G =$

Similarly we can do for windy, Humidity:-
We have IG for each feature.

IG (Y, Outlook)
IG (Y, Temp)
IG (Y, Windy)
IG (Y, Humidity)

→ max, pick the feature with most Informal gain and break the data set using that feature.

Suppose outlook gives most information gain, here we Breaked the data set using outlook.



At any point we have 3 cases
(i) pure-node ⇒ stop growing the node
(ii) cannot/good the best any more because of lack of points.
we stop growing when
(i) got a pure node
(ii) Node have very few points.
(iii) If we are too deep.

As depth of tree ↑ chance of overfitting ↑
If depth is ↓ " " underfit ↑
In DT, depth is the hyper parameter.

☆ Splitting numerical features:-
Till now we have only seen categorical feature Now, lets see how to deal with numerical feature.

f	y
2.2	1
2.6	1
3.5	0
3.8	0
4.6	1
5.3	0

① sort the numerical feature in ascending order.

② we check for each data point as splitting criteria

f < 2.2 → Yes / No
f < 2.6 → Yes / No
f < 3.5 → Yes / No
f < 3.8 → Yes / No
f < 4.6 → Yes / No
f < 5.3 → Yes / No

→ max. Again Recursively do the same thing on smaller Dataset.

* Feature standardisation:-

Till now in each Algo(logistic Regr, SVM, etc) we have done feature standardisation. But in the case of Decision Tree we are not computing distance i.e. it is not a distance based algorithm. Here we don't need feature standardization. because we don't care about scales here.

* Categorical features with many categories:-

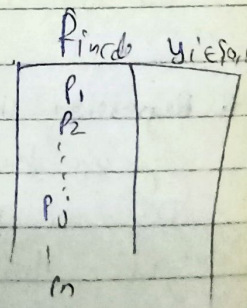
suppose we have a feature pincode, it is a numerical feature but really it is a categorical feature. It may have 1000's of categories. There may be some categories which may contain very few points.

we can apply feature engineering:-

pincode/zipcode ⇒ categorical

↓

convert it to numerical feature.



So how do we convert it to numerical feature.

$P(y_i = 1 \mid P_j)$

(is probability of $y_i = 1$ given my pincode j

$$= \frac{\#(y_i = 1 \;\&\; P_j)}{\#P_j}$$

Now, we can converting every pincode j into numerical feature which is equivalent to probability of $y_i = 1$ given the pincode j

Now, converting pincode to probability is very beneficial. we can treat them as normal numerical feature

* Overfitting & underfitting :-

As depth ↑ :- possibility of having very few points @ a leaf node increases and hence they may be noisy points and we are attempting to correctly classify them and here overfitting happens.

As depth increase interpretability decreases.

① if depth is very low then we are underfitting.

The correct depth can be predicted using cross validation.

* Time & space complexity

Train :- Time : $O(n(\log n)d)$ → At every stage we need to evaluate each feature IG.
n = # points
d = dim
↳ sorting of numerical feature

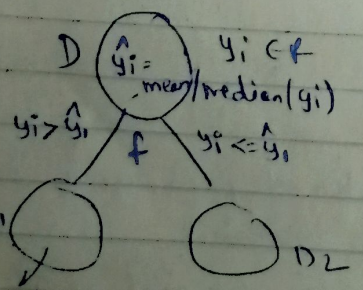
Testing: Time :- $O(D)$ → Depth of tree.

Space :- O(no. of nodes in tree)

∴ DT are good for, large data, dim is small, low latency

* Regression using DT :-

We have seen DT as a classifier. We can extend DT as Regression. We don't use IG here, instead of that we use mean squar error or MAD to break the data.



We ~~use~~ ~~more~~ chose the feature with lowered MSE or MAD.

Cases:

(i) Imbalanced data :- balance it using upsampling or downsampling because imbalance data impact entropy calculation

(ii) Large dim :- @ each node we have to evaluate each feature
Time complexity increases
avoid one-hot encoding for categorical feature because suppose feature have 10 categories Ho 1 feature will be converted to 10 feature :-
If feature have lot of categories it's useful to convert them to numerical feature of probability

(iii) Similarity matrix :- DT cannot handle similarity matrix ; they need data points

(iv) Multi-class clas-:- DT can handle multi-class . . If there are more than one type of points at a node choose the majority one

(v) decision surface :- Axis parallel types cuboids.

(vi) feature interaction :- logical feature interaction is inbuilt in DT like If outlook==sunny & humidity==high then play=No (feature interaction)

(vii) Outlier :- if depth is more ; outlie impact the DT.

(viii) Interpretability :- DT are very much interpretable.

(ix) feature importance :- We can do feature importance by calculating the reduction in entropy at each node - Sum up all the reduction due to a feature , if sum is large then it is more importance.

CH- Ensemble models

→ Ensembles :- In english ensemble means group of things. In ML ensemble means multiple model & combined together to build a powerful model.

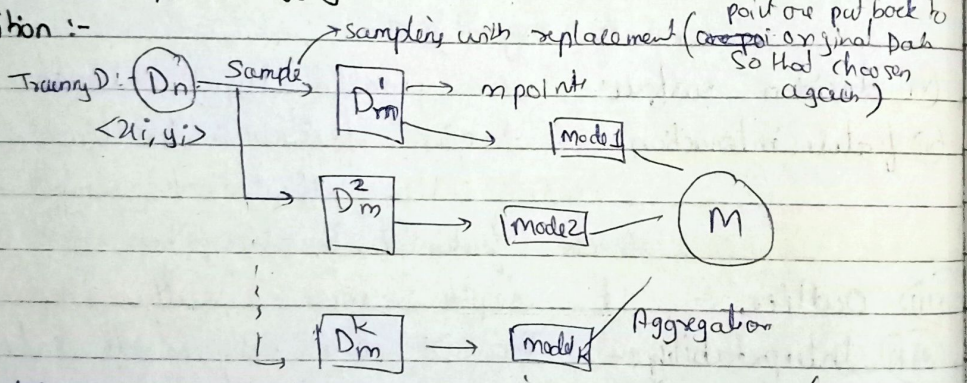
There are 4 types of Ensembles

1. Bagging (Bootstrapped Aggregation)
2. Boosting
3. Stacking
4. Cascading

Key output :- more different the models are, the better you can combine them.

(1) Bagging (Bootstrapped Aggregation)

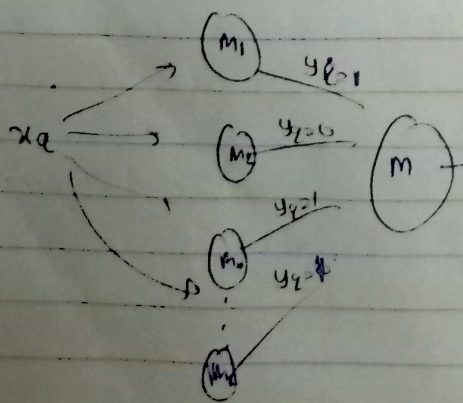
* Intuition :-



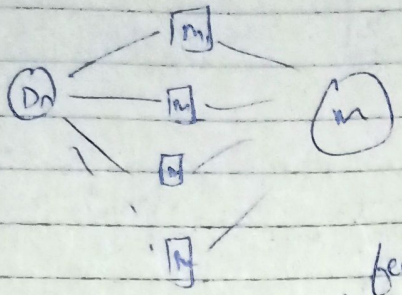
Each model m_i is build using D'_m of size m ($m \leq n$)
⇒ each model trained on different subset of data.

A typical way of Aggregation is in classification :- Majority vote
in Regression :- Mean/median.

Ex :-



* Bagging can reduce variance in model. Let see how variance is → how your model change as the data set change.



Each of the model are not seeing the whole data, they are train only on small subset of data. If the Data set is changed then only few of the Model will change and hence there is less chances that our final output after aggregation get change. Hence even for changes in Dataset final output may not change and hence variance decreases.

∴ Bagging can reduce the variance in model without impacting the Bias.

Now suppose each of our M_i is low Bias high variance model.

*** Bagging (M_i's) ⇒ low bias, reduced variance model.

∴ Bagging says take a bunch of low bias & high variance model & combine them using Bagging and you get a low bias and reduced variance model.

Ex:- of a low bias, high variance model is DT.

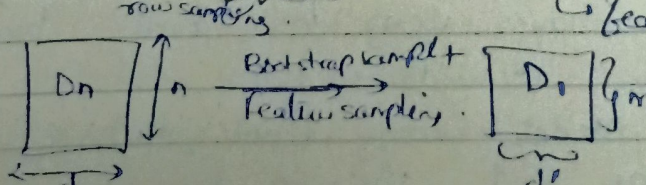
→ Random Forest:- It is the more popular Bagging model.

RF take DT as base model and apply bagging and also apply a unique tech called column sampling

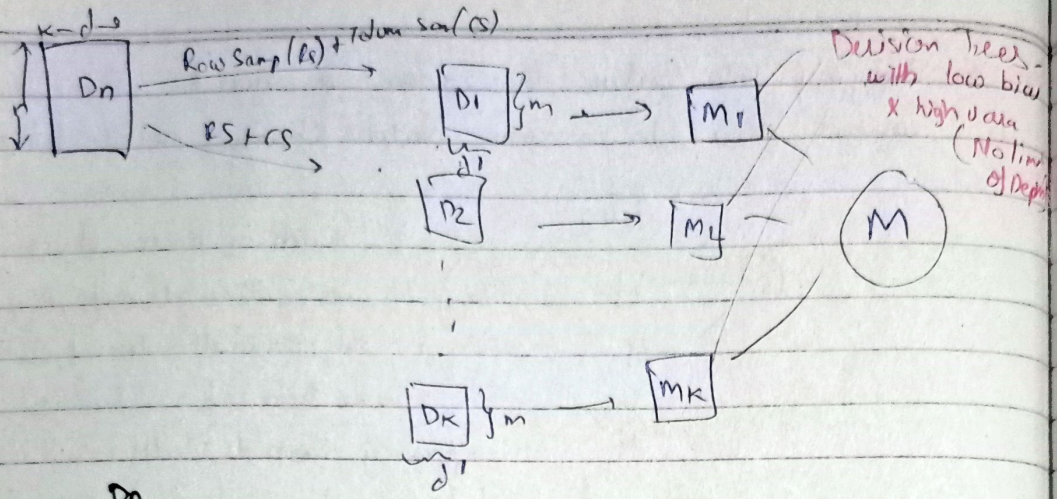
RF = DT + Bagging + col. sampling

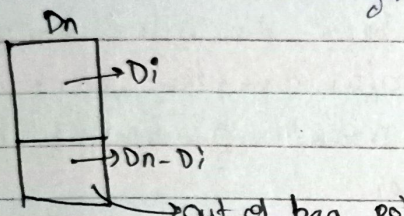
row sampling. feature bagging.

Train Data



Our idea of RF is, to sample both row & col^n.





These are not used to train M_i's here

They can be used for cross validation data for M_i

Ex:- Suppose for Model 1, we chose D_1 as D_i is subset of D_n then there must be some point which are not part of D_n those that points can be used as cross validate for Model 1. Same apply for all Model i

RF : DT + Row sampling + Colm sampling + Aggregate

- DT → fully grown Decision trees (i.e. No limit on height to make the model high variance)
- Row sampling → Bagging
- Colm sampling → feature sampling
- Aggregate → (Majority/Mean)

→ Bias - variance tradeoff :-

RF : have very low bias because base model (M_i's) are low bias.

Model = agg ($M_1, M_2, \ldots M_K$)

as k ↑ variance ↓

Mode no of models used in agg lesser will be the variance..

One more hyperparameter is diversity of samples.

$D_n \longrightarrow \underbrace{[D_1]}_{d'} \; m$ $\quad d' < d \quad m \leq m$

If these are small Dataset in each model will be more different hence variance ↓

~~Train~~ ✱ Time complexity :-

Train :- $O(n(\log n)\, d * K)$ $\quad n \rightarrow$ no. of points

But we can parallelize the training of base model's to reduce the time complexity.

When we have large amount of data with reasonable features. we can train more model parallely by created batches and parallely train models in batch :-

Test :- Time: $O(\text{depth} * K)$

Space: $O(DT * K)$

* Extremly Random Trees :- There is a variation of RF.

In RF :- row sampling, col^n sampling, aggregation

Extreme tree :- don't use all the values in a numerical feature to for decision making. As we have learned in DT that if a feature is numerical then we sort it and check for each value in that feature if it can ~~give~~ be a threshold. Instead of that Extreme tree only use random sample of possible values to determine threshold

∴ Extreme tree :- col^n sampling + r.s + agg + randomization when selecting threshold in numerical feature

Extreme tree tends to reduce variance better than RF but bias may increase.

(2) **Boosting:-** (base model) (Randomization)

In Bagging :- hig-variance, low-bias + (CS, RS) + aggregation

Boosting : low-variance, high-bias + additively combine.

reduce bias while keeping our variance low.

Core-idea :- how boosting reduces bias

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

Step 0:-(a) Build a model M_0 on whole of D_{train}.

M_0 need to have high bias low variance

$M_0 \leftarrow \{x_i, y_i\}$ for example a DT with small depth.

$h_0(x)$ As we have high bias model we ~~do~~ encounter large training error.

(b) calculate $error_i = y_i - \hat{y}_i$ {Regression}

∴ ~~Each~~ point in train data x_i, y_i we have err_i

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

Step 1 :- train model m_1 with take $\{x_i, error_i\}$ as its training data.

$m_1 \leftarrow \{x_i, error_i\}_{i=1}^{n}$

$h_1(x)$

$f_1(x)$ = model at end of step 1.

$f_1(x) = \alpha_0 h_0(x) + \alpha_1 h_1(x)$

step2 :- $\{x_i, err_i\}$

$\hookrightarrow y_i - f_1(x_i)$

$f_2(x) = \alpha_0 h_0(x) + \alpha_1 h_1(x) + \alpha_2 h_2(x)$

⋮

Step k :- $F_k(x) = \sum_{i=0}^{k} \alpha_i h_i(x)$

additive weighted model.

Each model is trained to fit the residual error @ end of the prev-stage.

$$F_K(x) = \sum_{i=1}^{K} \alpha_i h_i(x)$$

↓

ends up having a low residual error

∴ Train error ~~do~~ reduces bcaus at each step we try to fit the residual errs & and here bias reduces.

There are many boosting algorithm some of them are

(i) Gradient Boosted DT (GBDT)

(ii) AdaBoosting (Adaptive boosting)

* Residuals, Loss-funcn & Gradients:-

$$F_K(x) = \sum_{i=1}^{K} \alpha_i h_i(x)$$

residuals:- residual at end of stage K is

$$\underbrace{err_i}_{\text{residual}} = y_i - F_K(x)$$

Loss-funcn:- $L(y_i, F_K(x)) = (y_i - F_K(x_i))^2$

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

$(y_i - F_K(x_i))$ → residual.

$$\frac{\partial L}{\partial F_K(x)} = -2 * \text{residual}$$

$$-\frac{\partial L}{\partial F_K(x)} = 2 * \text{residual}$$

negative derivate/gradient

* negative gradient ≈ residual.

of loss funcn wr.t $(F_K(x))$.

↑ pseudo residual

Now instead of using residual we can train of pseud-residual.

The advantage. pseudo-residual helps us to chose any loss funcn. as long as the loss fun is differentiable

for example RF cannot be used to min hinge loss.
But gradient boosting can be used to minimize any loss
That's a big advantage.
So algorithm now looks like this
At any stage i : we have already built a model $F_{i-1}(x)$

$(m_i) \leftarrow (x_i, err_i)$
↳ pseudo-residual.
$-\frac{\partial L}{\partial F_{i-1}(x)}$

Gradient Boosting :-

Algorithm:

Ex { logistic loss
SQuared loss
Hinge Loss

Ip :- training set $\{x_i, y_i\}_{i=1}^{n}$, a differentiable loss funcn $L(y, F(x))$,
, number of iteration M ↳ # of base models ($m_1, m_2 \dots m_M$)

Algo :

1. Initialize model with a constant value :

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

// find a constant γ s.t error on training data is minimized

learn all model
↳ 2. for $m = 1$ to M :

1. Compute so-called pseudo-residuals :

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

2. Fit a base learner (e.g tree) $h_m(x)$ to pseud-residuals
i.e. train it using $\{x_i, r_{im}\}_{i=1}^{n}$

3. Compute multiplier γ_m by solving 1-Dim optimization problem

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

4. Update the model :

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

3. Output $F_m(x)$

So, what a base learner can be, the most popular learner is Decision Tree. We get Gradient Boosting Decision Tree. Decision Trees in GBDT have very small depth.

* Regularization by shrinkage:-

So the formulation of Gradient Boosting

$$F_m(x) = h_0(x) + \sum_{i=1}^{M} \gamma_m h_m(x)$$

m = # base-models

as # base-models ↑ , variance ↑ & has overfit ↑

But there is a concept called shrinkage.

An important part of gradient boosting methods is regularized by shrinkage which consists in modifying the update rule as follows:

$$F_m(x) = F_{m-1}(x) + \nu \cdot \gamma_m h_m(x) \qquad 0 \leq \nu \leq 1$$

where ν is called learning rate.

Empirically it has been found that small learning rates (such as $\nu < 0.1$) yields dramatic improvement in model's generalization ability over gradient boosting without shrinking ($\nu = 1$). However, it comes at the price of increasing computational time both during training & querying: lower learning rate req. more iterations.

So we have 2 hyper parameters M & ν.

* Time complexity:-

Train : Time: $O(n (\log n) d * M)$

But Remember, Remember, RF was trivially parallelizable but GBDT is not easy to parallelize. Hey, this is a serial algo.

Runtime : Time:- $O(\text{depth} * M)$

↳ small in GB (because in GBDT we take DT with very small depth)

Space:- $O(DT * M + \gamma_m)$

* XGBoost + Randomization:-

XGBoost is an implementation of GBDT, in its implementation, in addition to normal GBDT, it also includes Random Sampling & Column sampling.

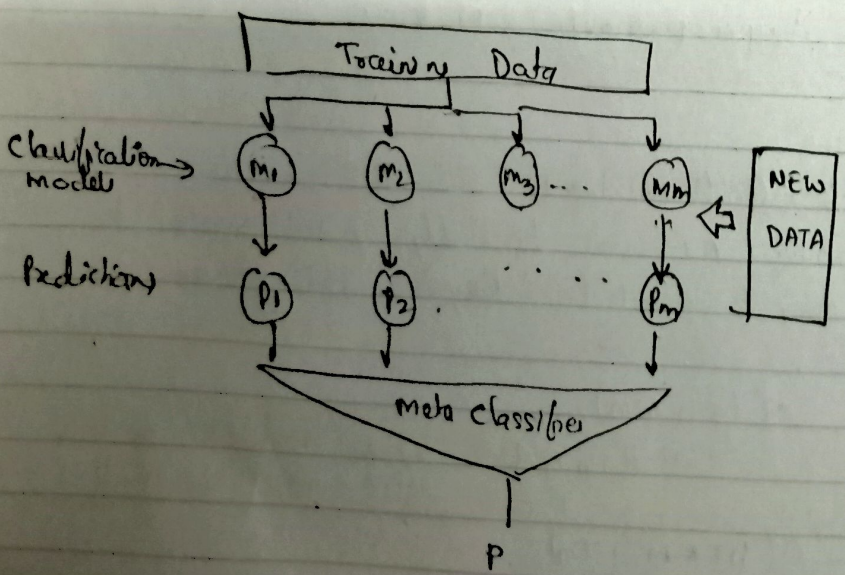
XGBoost implementation is faster than sklearn implemen of GBDT.

It is developed by phd students of university of washington.

* Ada Boost :- It is similar to Gradient Boost. It is mostly used in image detection. In case of GBDT we used pseudo residual. Here all the miss classified points are assigned more weight.

③ stacking classifier:-

Stacking is an ensemble learning technique to combine multiple classification models via a meta-classifier. The individual classification models are trained based on the complete training set, the meta-classifier is fitted based on the o/p - meta features - of the individual classification models in ensemb The meta classifier can either be trained on the predicted class labels or probabilities from the ensemble.



This is same as RF but the diff. is, in RF each base-model is DT but in stacking each of the model may be different more different the models are more good the working is. And in RF we have sampled the data but here we create various types of model on whole training data.

Algo: Input: $D = \{x_i, y_i\}_{i=1}^{m}$ $(x_i \in R^n)$

O/p: An ensemble classifier H.

Step 1: Learn the first level classifier

for $t \leftarrow 1$ to T do

Learn a base classifier h_t based on D

Step 2: Construct new data sets from D

for $i \leftarrow 1$ to m do

Construct a new data set contain $\{x_i', y_i\}$

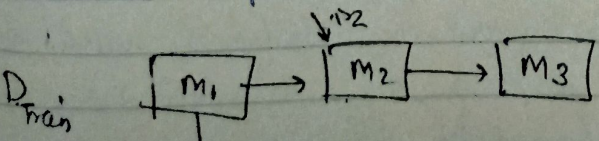
$x_i' = \{h_1(x_i), h_2(x_i), \ldots h_T(x_i)\}$

Step 3: Learn a second level classifier.

Learn a new classifier h' based on newly constructed data set.

return $H(x) = h'(h_1(x), h_2(x), h_3(x), \ldots h_T(x))$

① Cascading models:- They typically used when the cost of making a mistake is high i.e. we want a very very accurate prediction, we want to be 100% sure.



- $D_{train} \rightarrow M_1 \rightarrow D_1$
- $D_{train} - D_1 \rightarrow M_2 \rightarrow D_2$
- $D_{train} - D_1 - D_2 \rightarrow M_3 \rightarrow D_3$

We are reducing the training data, we are dealing the data which are correctly classified in previous model. It is mostly used in critical situation like credit fraud detection in bank and, medical diseases