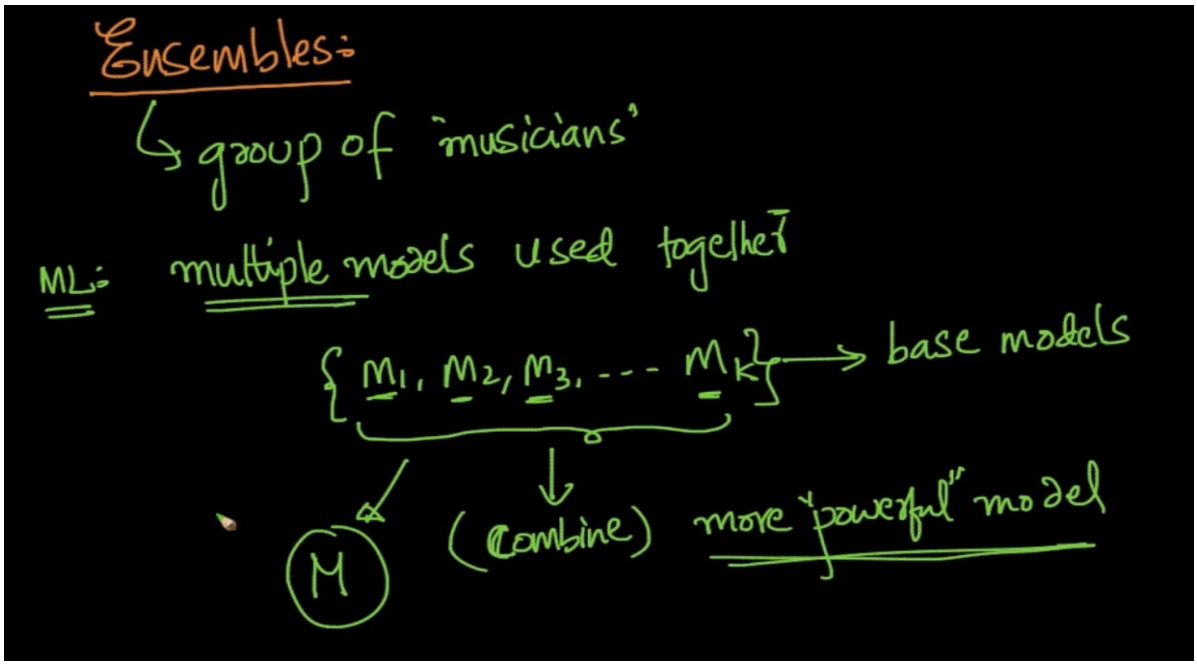
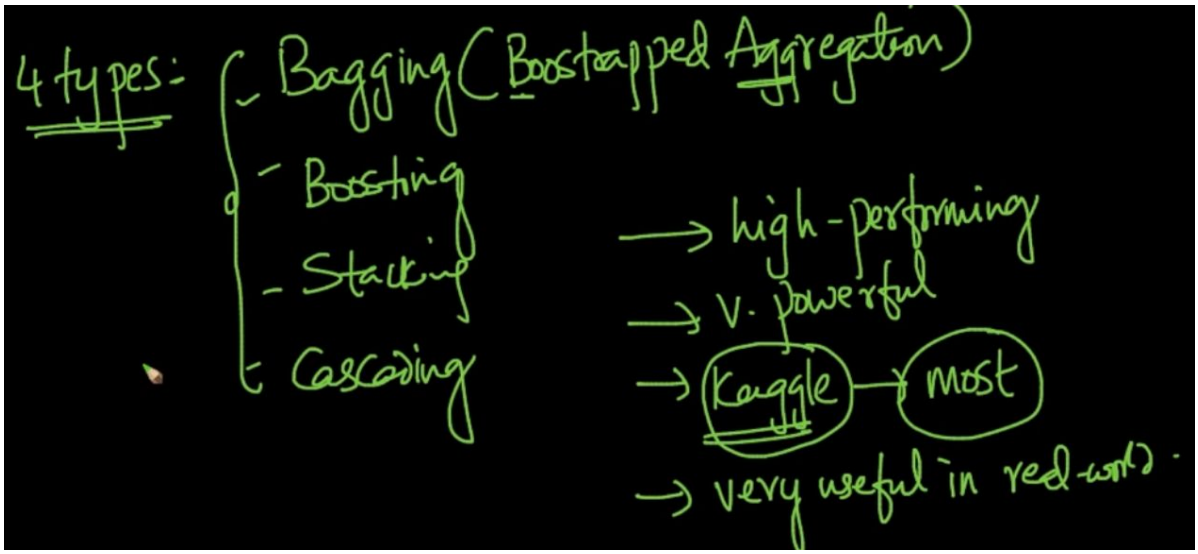
ENSEMBLE MODELS

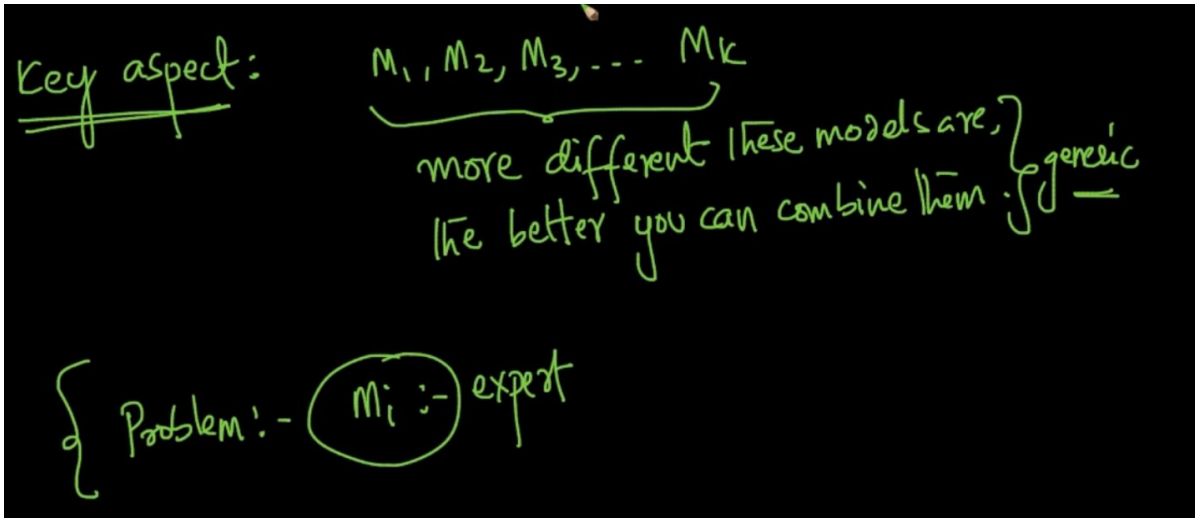
WHAT ARE ENSEMBLES ?



Ensembles : A group of Things . In the Machine Learning world it means using multiple models together i.e $\{M_1, M_2 ..... M_k\}$ these are base models like Logistic Regression, SVM and we combine them and try to take all the good properties of them and make a more powerful model 'M'

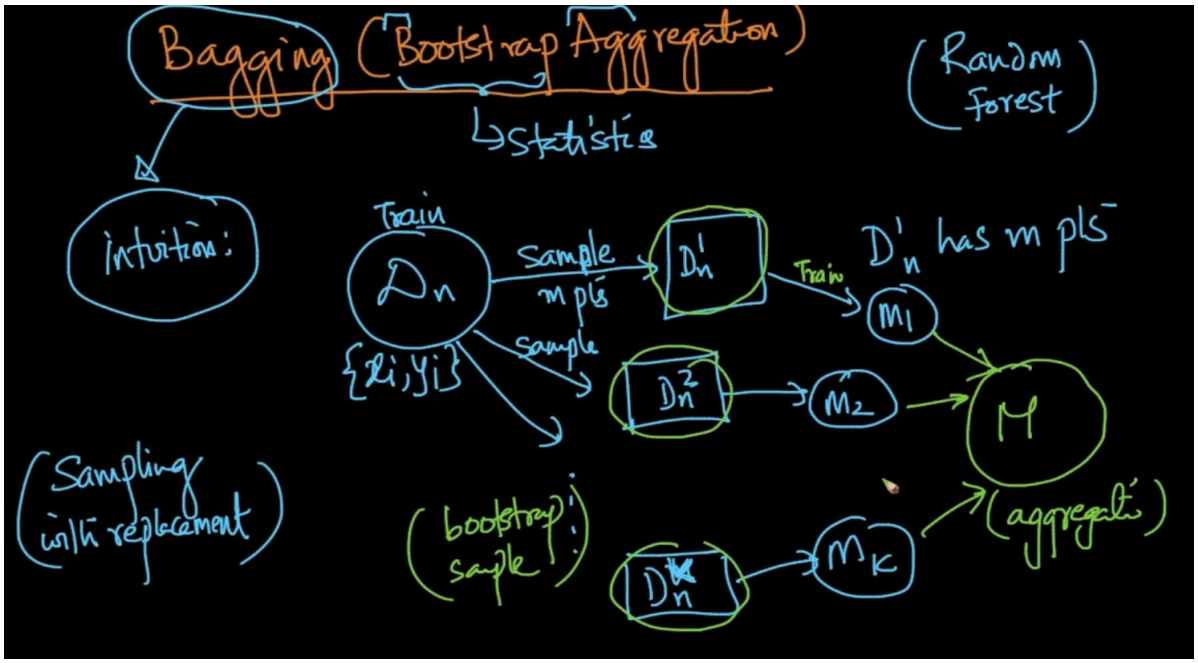


These are 4 types of Ensemble models

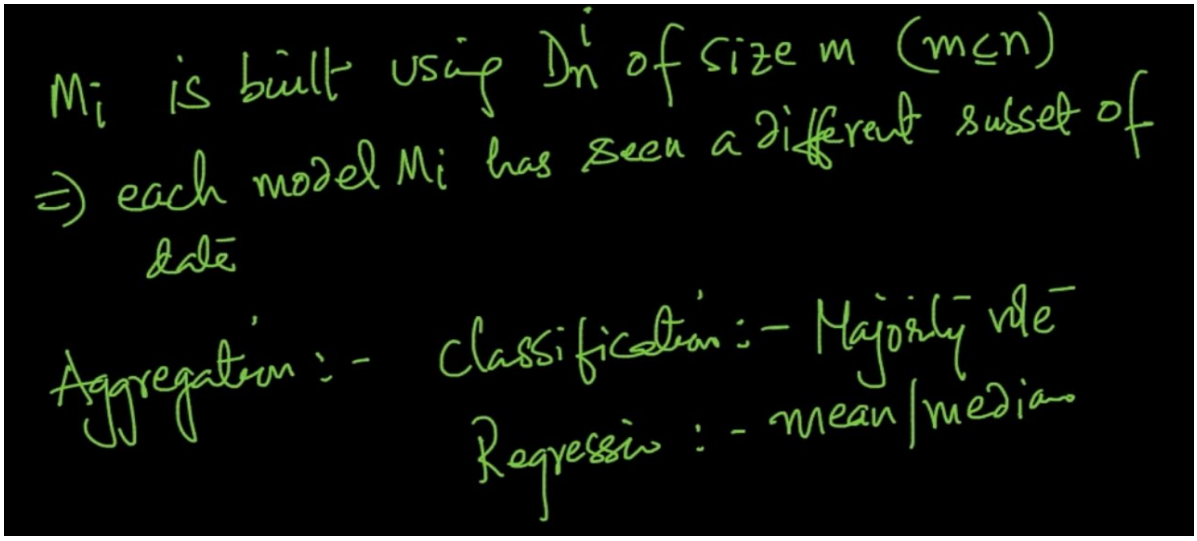


One of the key aspect is $\{M_1, M_2 ..... M_k\}$ the more different these models are the better you can combine them. Think of it as Experts of different fields and we are getting results using the combination of their opinions

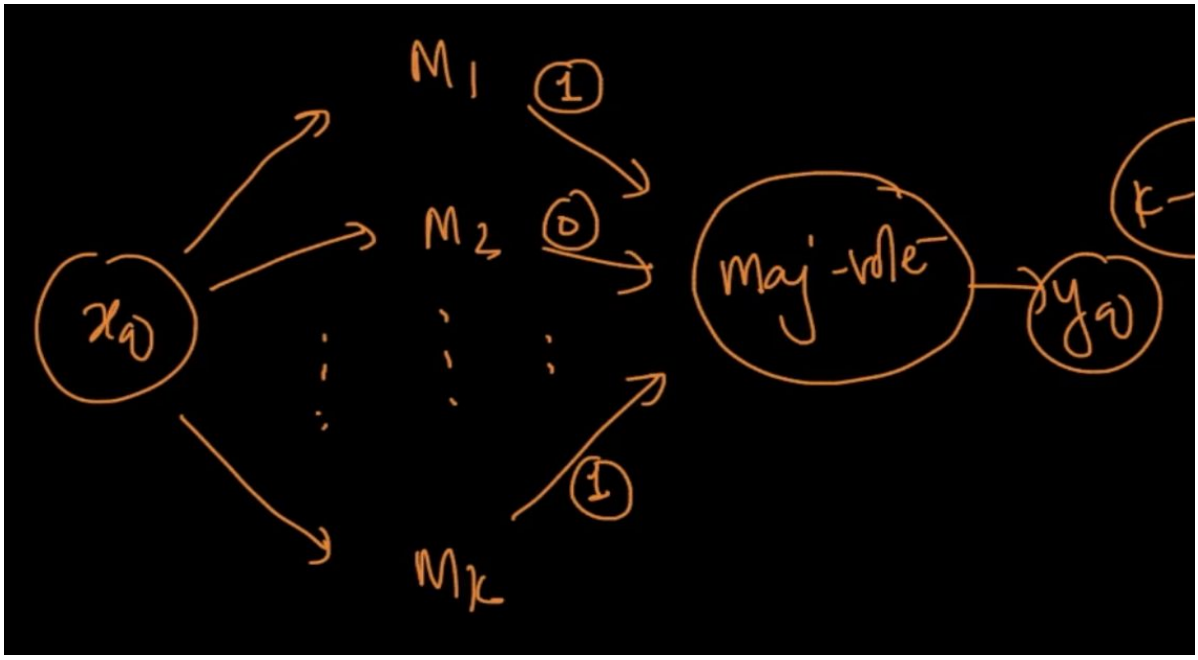
PART 1 : BOOTSTRAPPED AGGREGATION (BAGGING) INTUITION



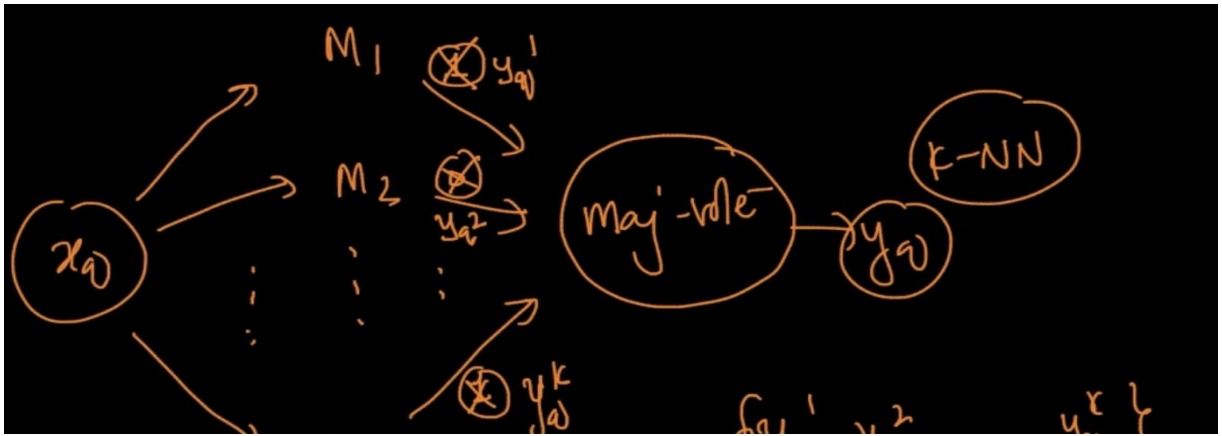
Bagging comes from B - Bootstrap, Agg - Aggregation . From our Train Data D_n we take samples and make a model from samples from D_n. Suppose we are taking 'k' samples all with m points and make k no. of models. Now we combine these models which is called aggregation of models and make a model M



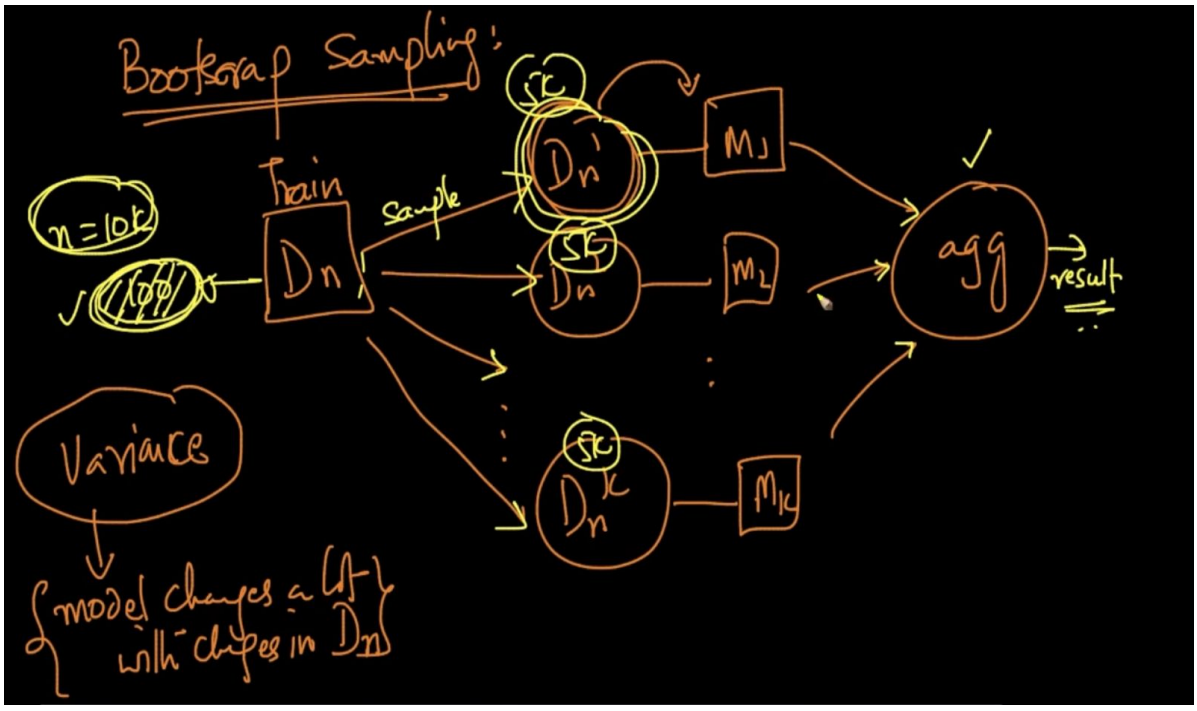
Each model sees a different subset of data.



Classification :- If a new query x_q comes we pass it to all the models and they predict labels i.e let's say model M_1 => 1, M_2 => 0..... M_k => 1 and if majority gives 1 then y_q = 1



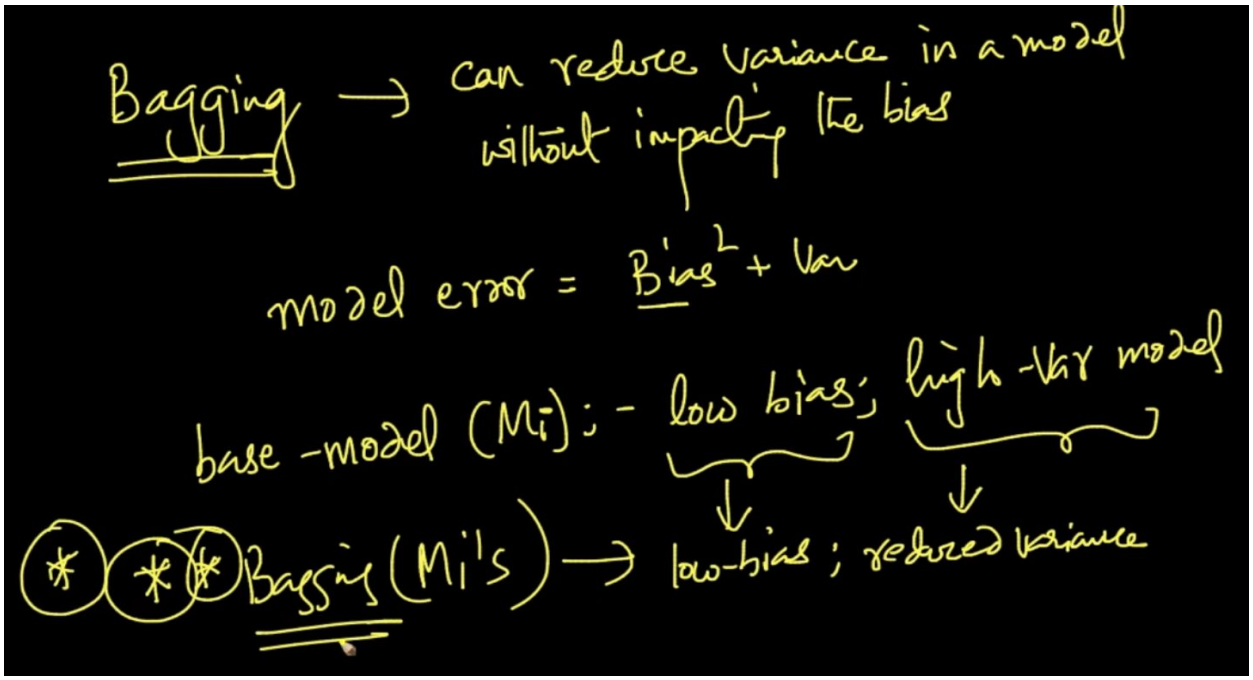
Regression :- If a new query x_q comes we pass it to all the models and they predict labels i.e let's say model M_1 => y_q^1, M_2 => y_q^2 ..... M_k => y_q^k and we store $\{y_q^1, y_q^2, ..., y_q^k\}$ then we take the mean of all the predicted y_q. So, y_q = mean $\{y_q^i\}$



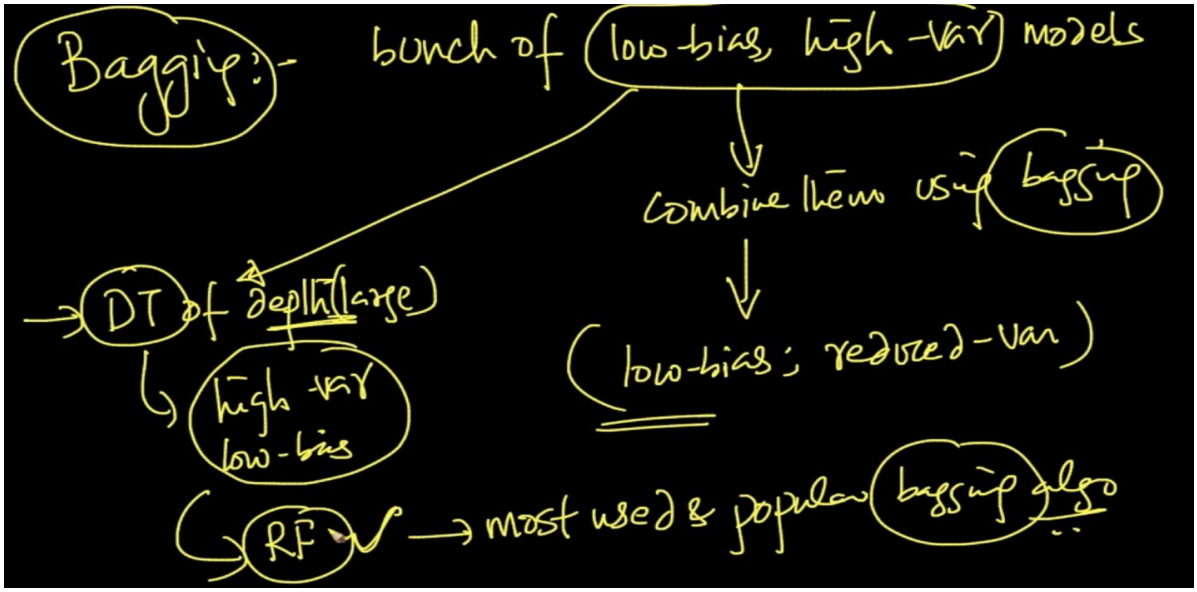
We've n = 10k points. We take 5k points randomly in our samples for creating models

We say Variance of a model is high if model changes a lot in with changes in Train Data.

So if 100 points are changed then in our Bagging pipeline not all the samples will have these 100 changed points, only some so with majority vote our final result isn't getting impacted much. This is the core idea of Bagging

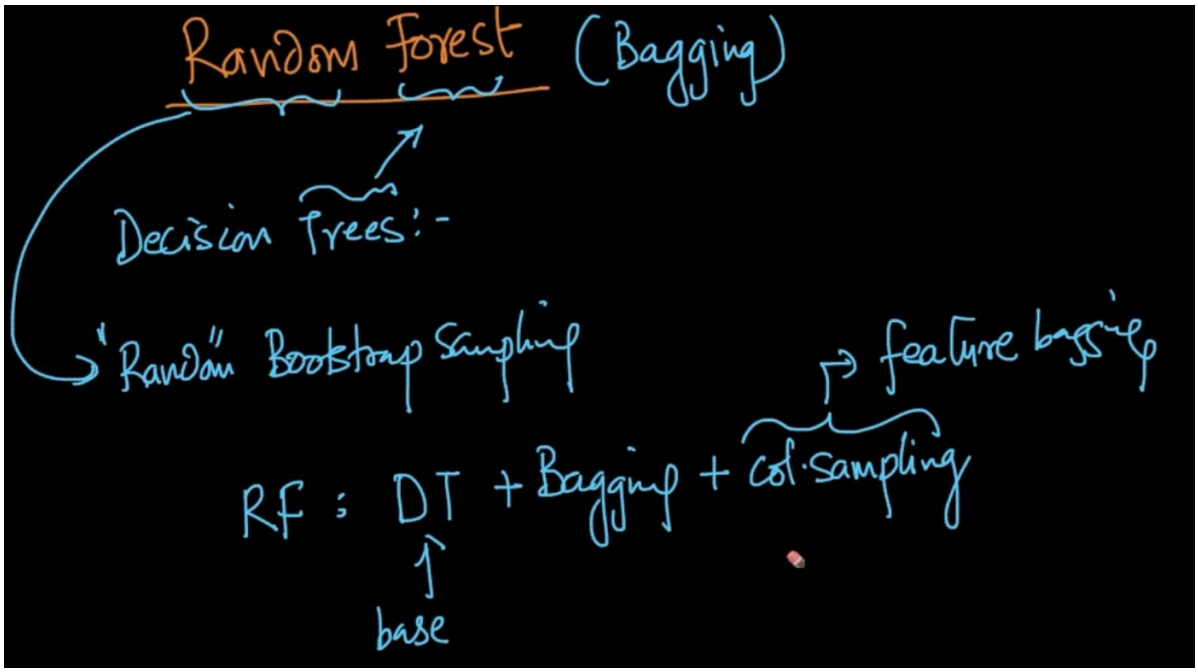


The above theory implies that Bagging can reduce variance in a model without impacting bias. Model Error : $Bias^2 + Variance$. If our Base models (M_i) are low bias, high variance models then with Bagging on $M_i's$ we'll still have low bias and also reduced variance



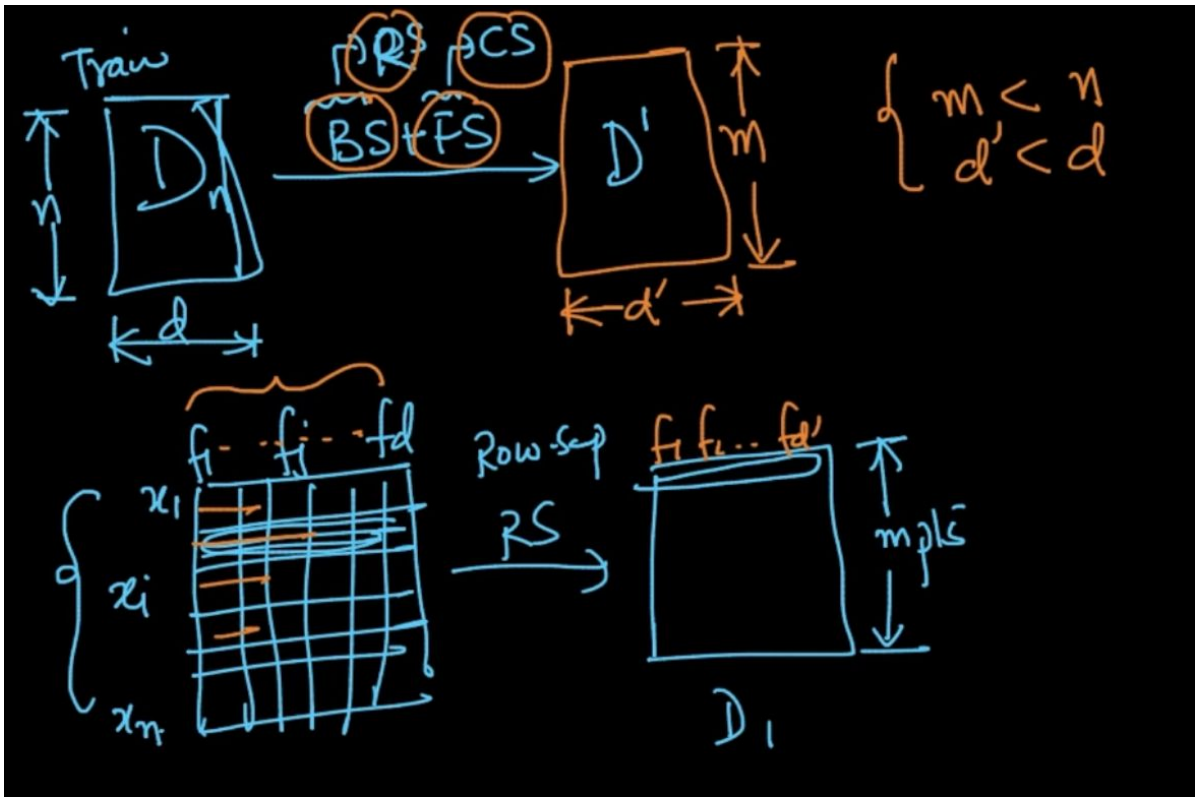
Decision Trees with large depth have high variance , low bias so when we apply Bagging on these models and combine them . It's called Random Forests (RF) which is the most popular Bagging algorithm

RANDOM FOREST AND THEIR CONSTRUCTION



Random Forests : “Random” comes from Random Bootstrap Sampling that we do in Bagging and “Forest” comes from Trees in Decision Trees

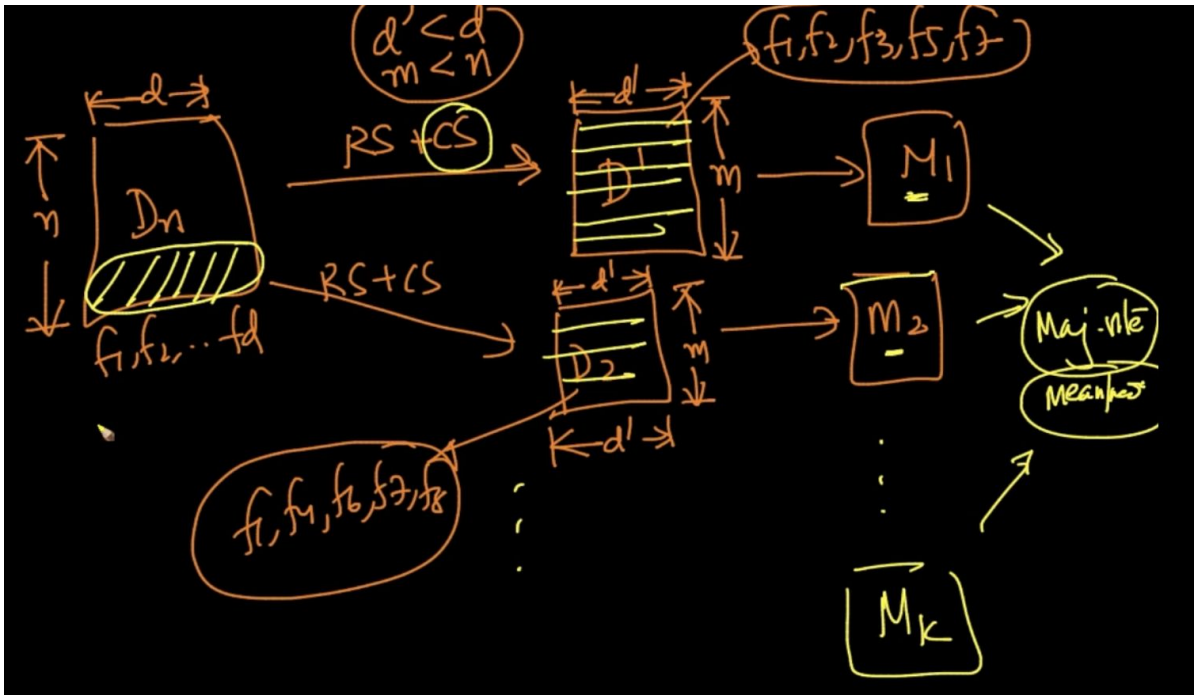
Random Forest = Decision Trees (as Base models) + Column Sampling (Feature Bagging)



Row Sampling/ Bootstrap Sampling : Taking random points $x_i's$ from our dataset. If we've n points and our Bootstrapped Sampling has m points then m < n

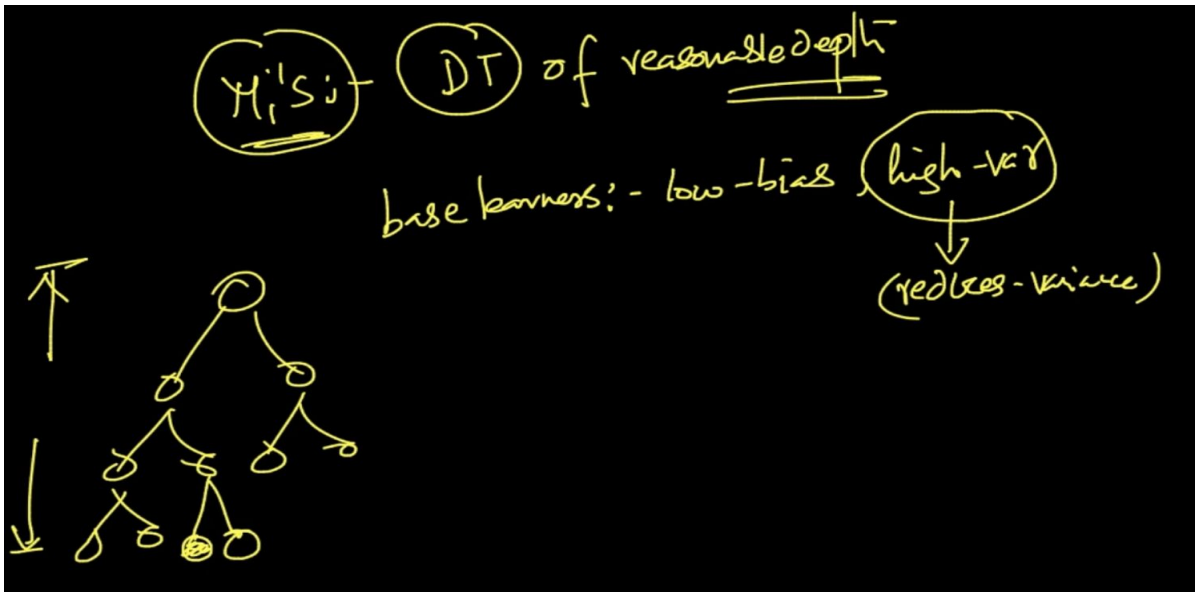
Column Sampling/ Feature Sampling : Taking random features from all our features. We don't select all the features but some features randomly. If we've d dimensions in our actual dataset and d' dimensions in sample then d' < d

So we apply both these Sampling (Row Sampling + Column Sampling) in Random Forest

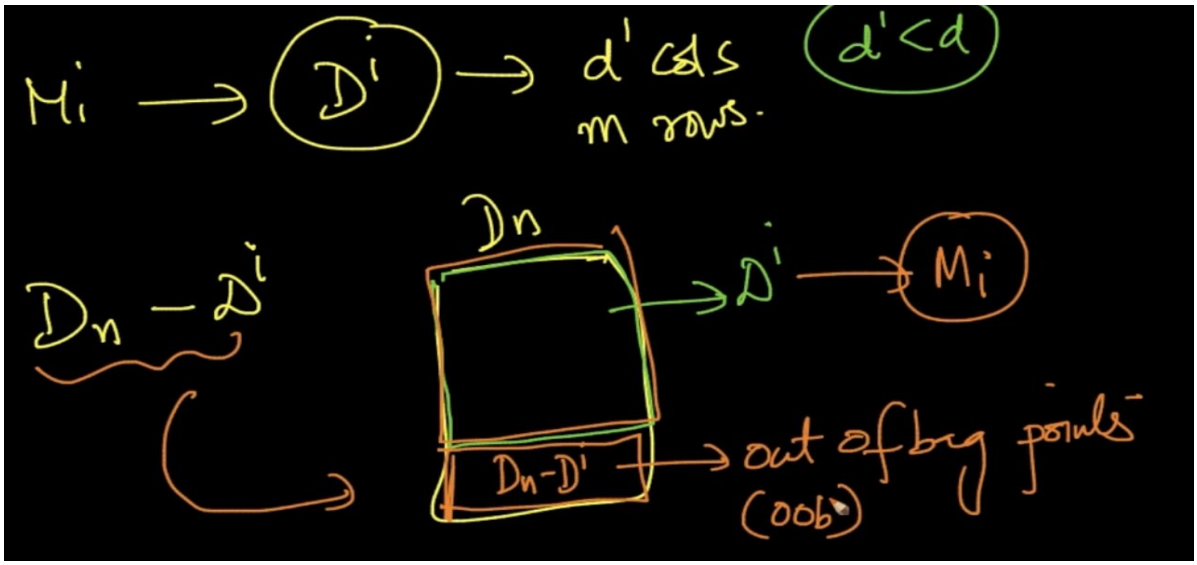


As it can be seen in D_1 we create model M_1 with m points and d' features by applying Row Sampling + Column Sampling. Same for D_2 and further models

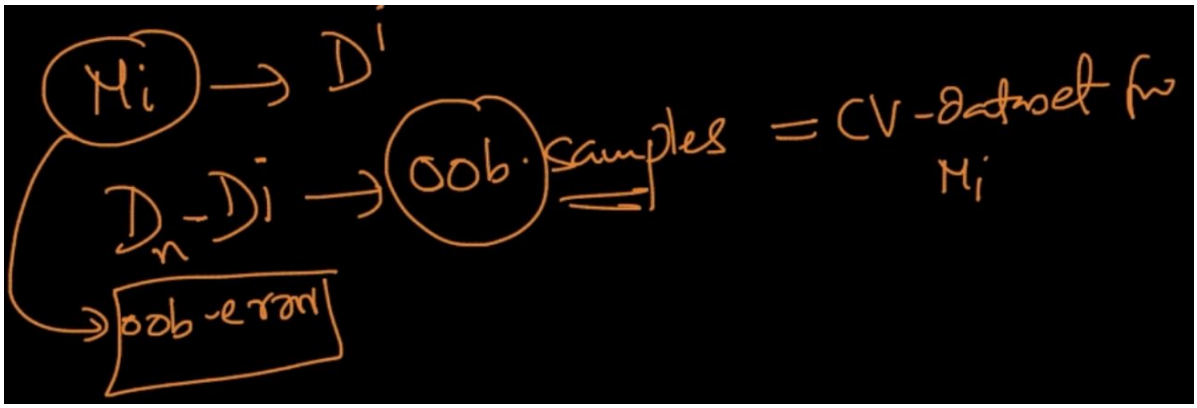
Note that in D_1 and D_2 both have different d' as D_1 has d' = $\{f_1, f_2, f_3, f_5, f_7\}$ and D_2 has d' = $\{f_1, f_4, f_6, f_7, f_8\}$ and m points are also there but they are also not same but randomly chosen. So both models M_1 and M_2 are very different and we take Majority Vote by combining results of all the models



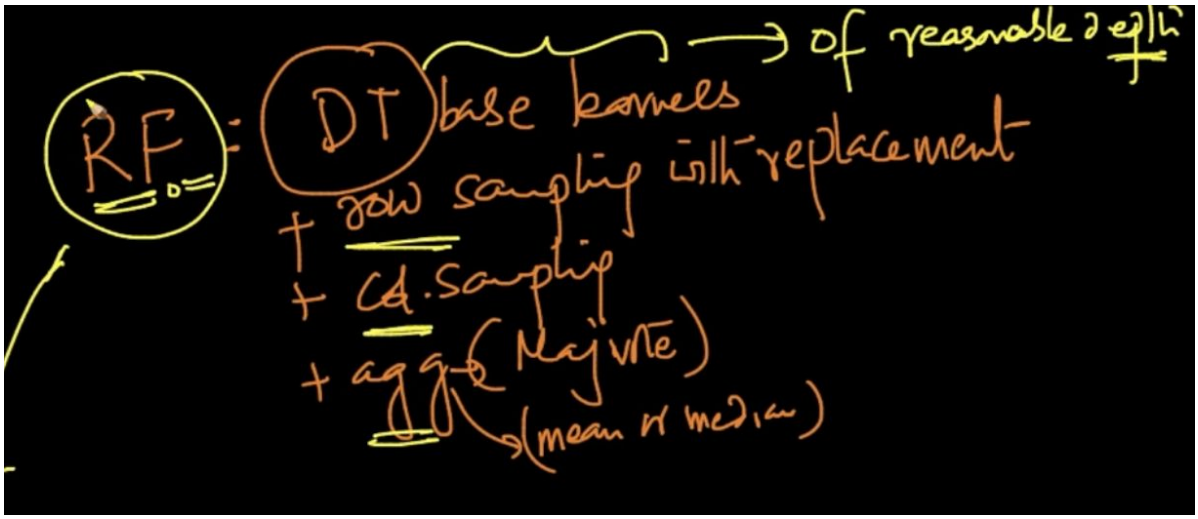
All our Base models ($M_i's$) are Decision Trees of reasonable depth. We'll have no problem if they've low bias and high variance since Aggregation function reduces Variance. We can use Decision Trees even if they've high depth or Overfitting because it means high variance which we can deal with.



We want our models M_i to be sensible . For that we do Cross Validation. We need CV data for that. Our model M_i is created from D_i .So, $D_n - D_i$ which are also known as out of bag points can be used as CV data

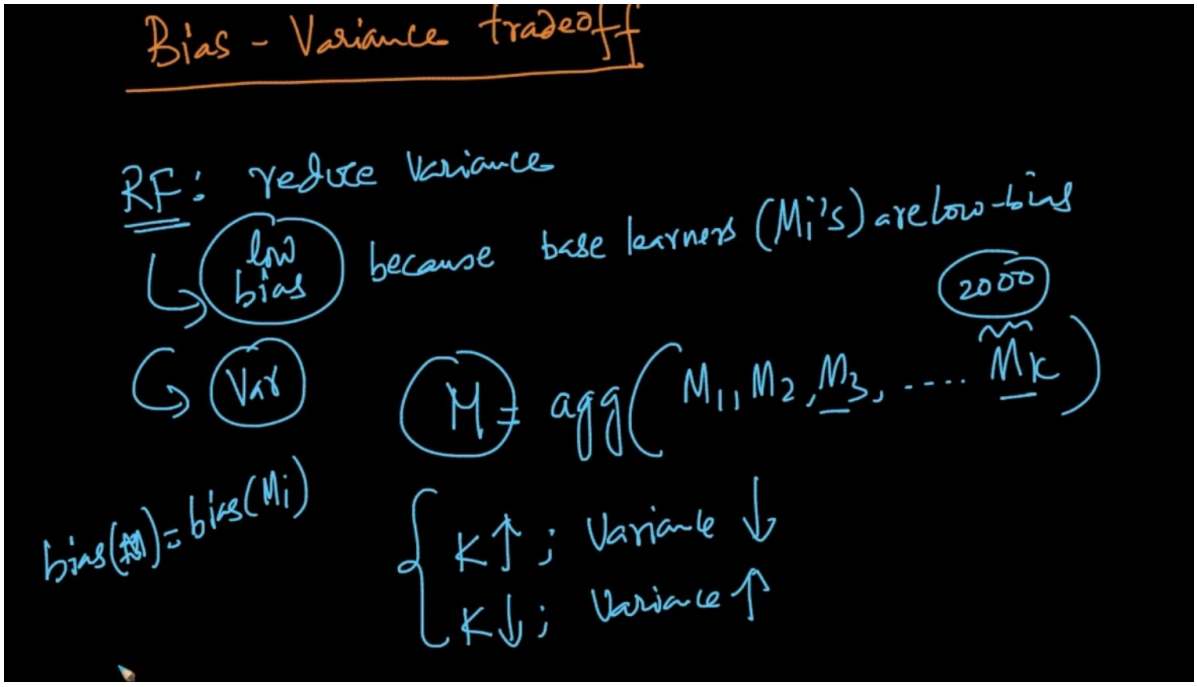


So lot of libraries have oob-error as a parameter

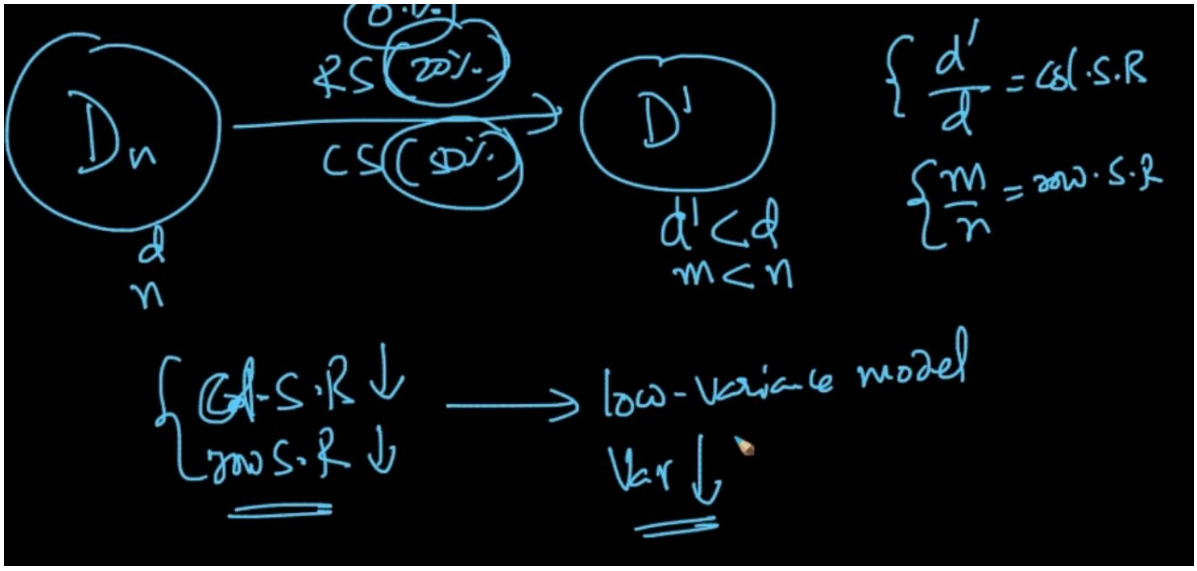


Random Forest : Decision Trees (Base Learners) of good depth + Row Sampling with replacement + Column Sampling + Aggregation (Majority vote/Mean or Median)

BIAS - VARIANCE TRADEOFF

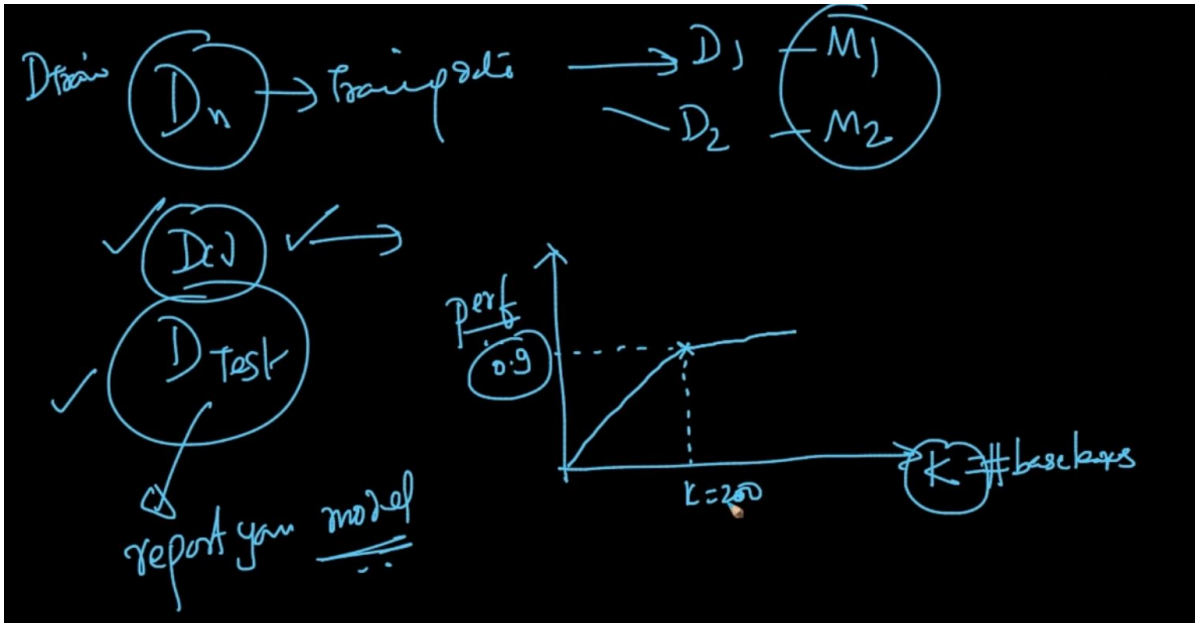


Random Forests have low bias because base learners (M_i) have low bias. As no. of models increase the variance decreases and vice versa.



Column Sampling Rate (C.S.R) = $\frac{d'}{d}$ where d' - Dimensions in Sampled columns, d- Total dimensions. Row Sampling Rate (R. S. R) = $\frac{m}{n}$ where m - Points in sample, n - Total points

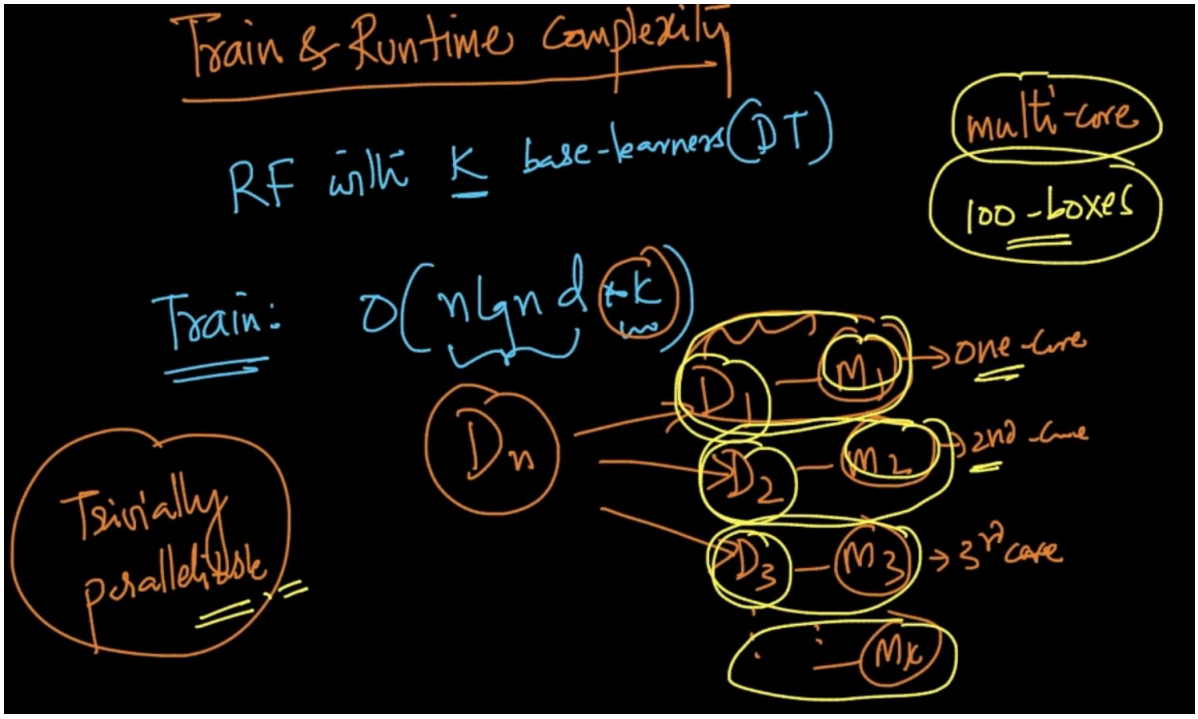
When C.S.R and R.S.R are low we can have low variance model because each sample will have different Columns and Rows so they won't be much co-related. We can fix RSR and CSR but we need to keep in mind that it isn't too small



The most general practice is We use the Train Data , create Samples and create models

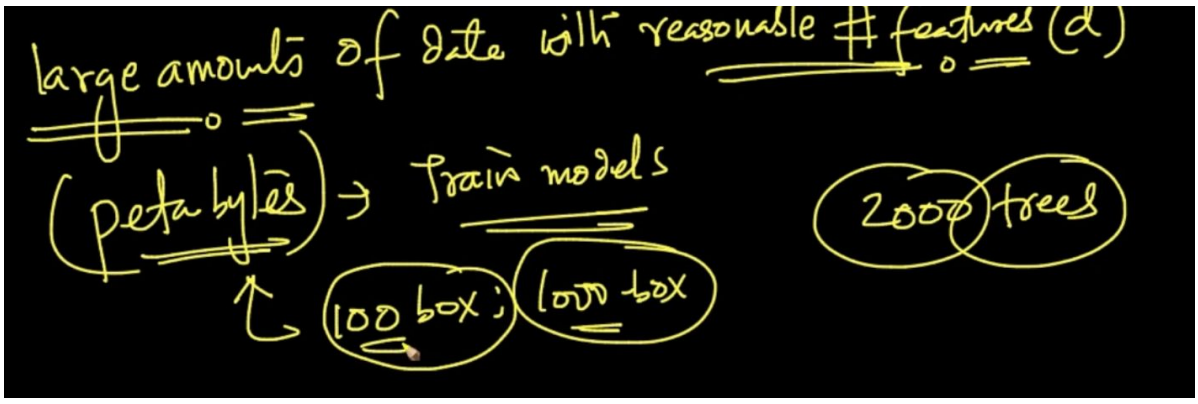
Our cross-validation data D_{CV} is used to decide the 'k' - No. of base learners. It's the most important parameter in Random Forests. We can decide the k using D_{CV} . k also helps in reducing variance as k ↑ , Variance ↓

RUN-TIME AND TRAIN COMPLEXITY

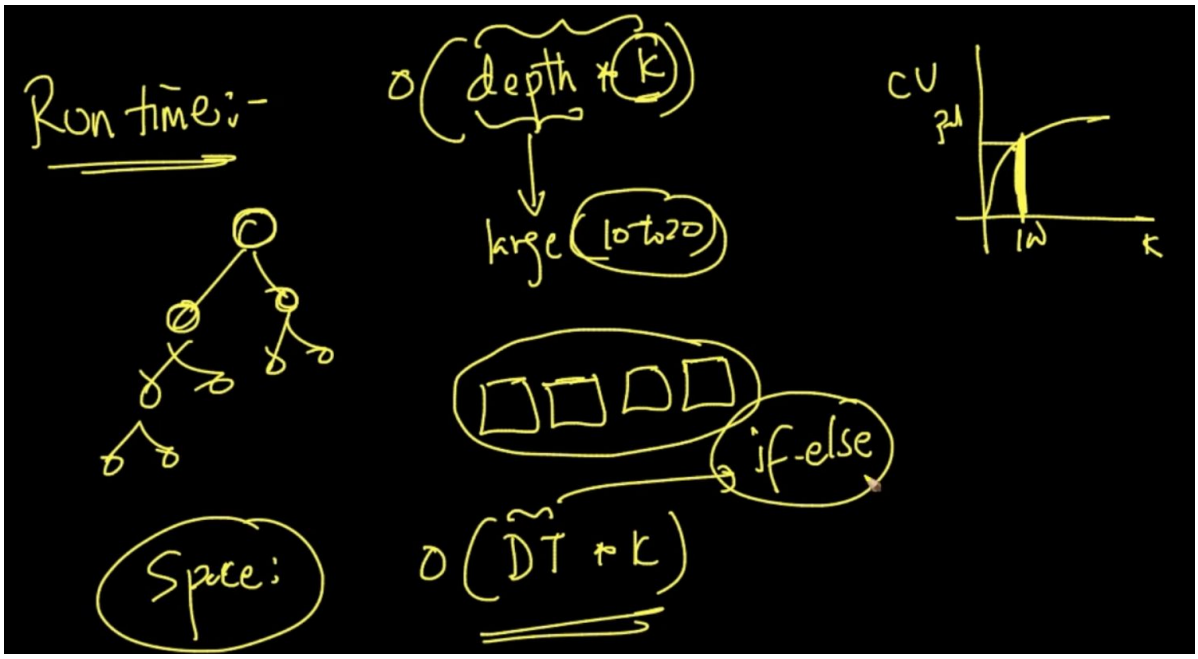


Train Complexity : O(n*lg(n) * d * k) where d-Features/Dimensions , k - No. of Base Learners. The amazing thing abour Random Forest is that they can be trained parallely

We create base models using samples as the samples are not dependent/linked with each other. We can train 1 model on 1 core and 2nd model on the another and so on

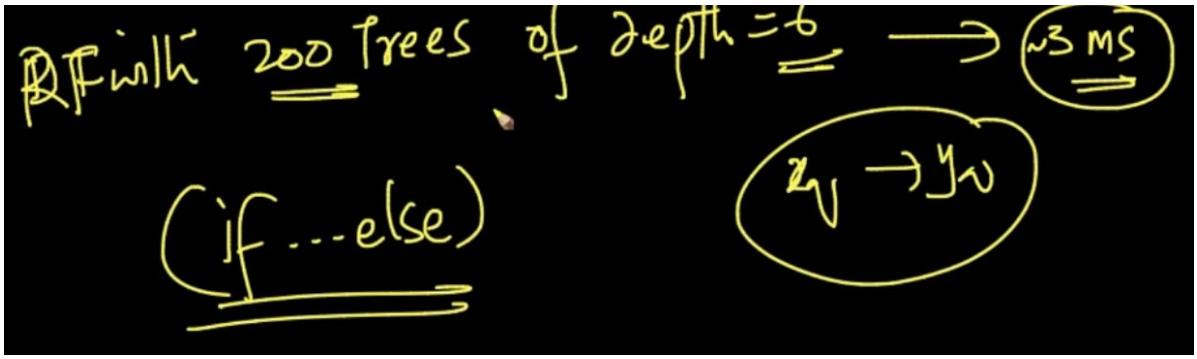


If we've large amounts of data then 100/ 1000 machines can be used parallely to train models. We can work with as large as 2000 Trees as well with this



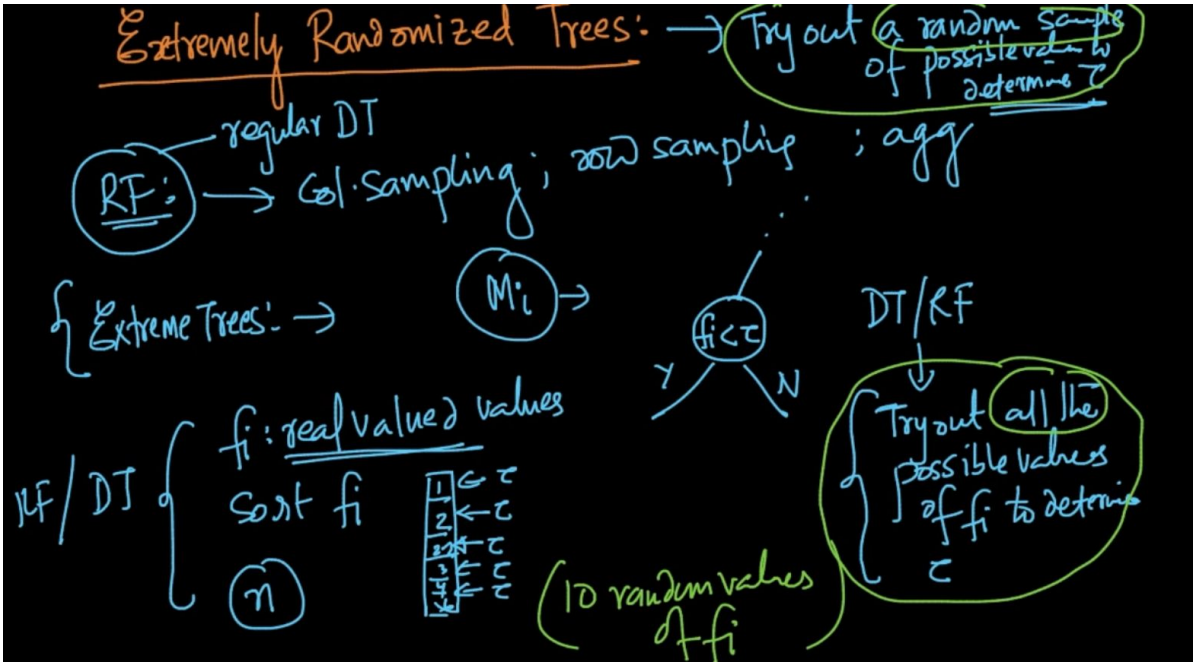
Run-time complexity : O(depth * k) which isn't too much

Space Complexity : O(Decision Trees * k) . Decision Tree has to be stored which isn't much as it's just if-else conditions which are optimized by modern day compilers



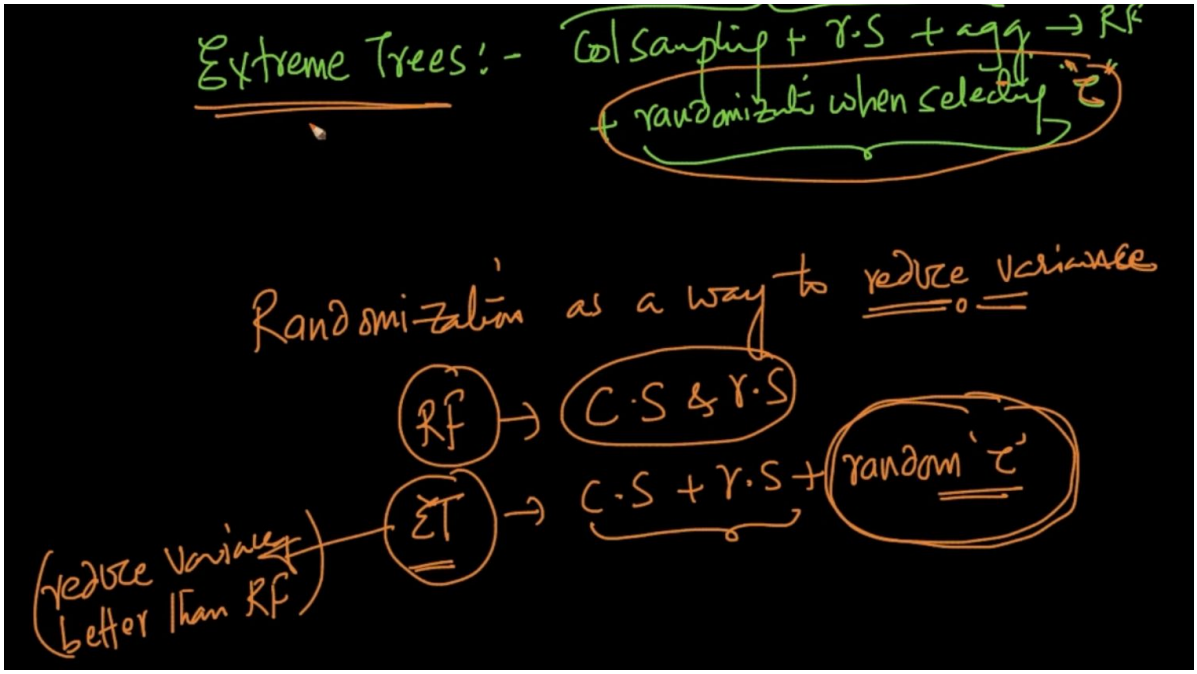
We can easily get result from a Random Forest with 200 trees of depth = 6 in 3 ms

EXTREMELY RANDOMIZED TREES



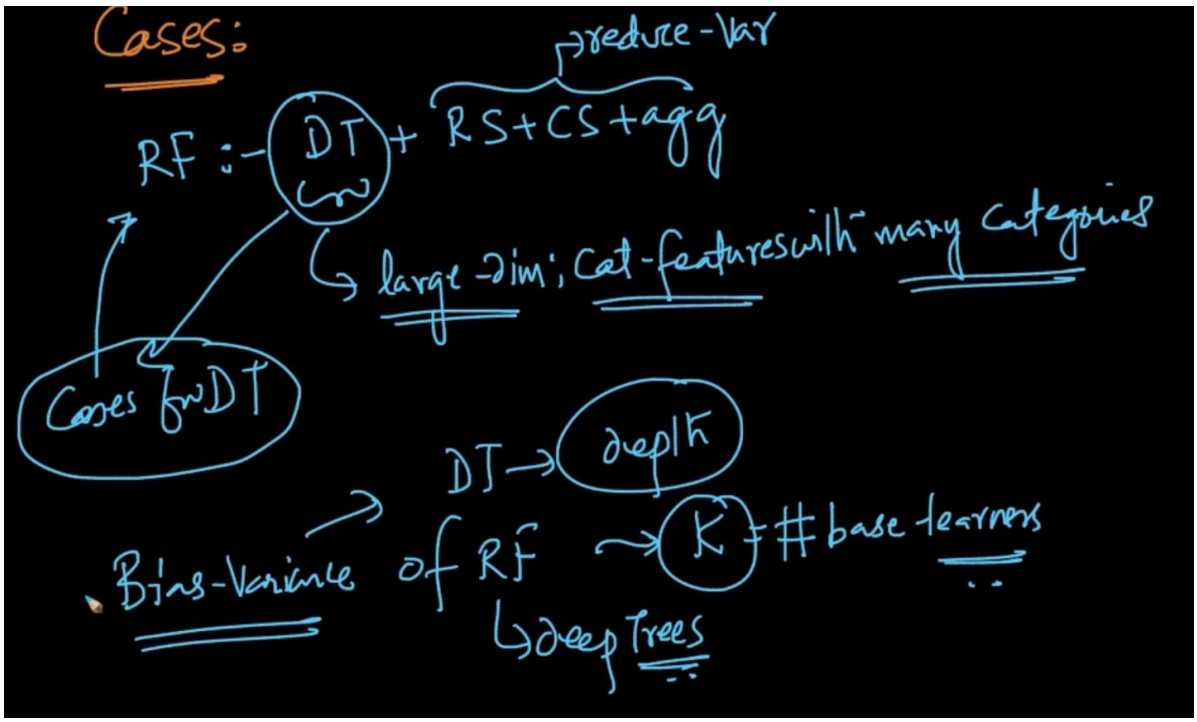
We do the above things in Random Forest but in Extreme Trees we take a step further in addition to that. Let's see how we do that

Extreme Trees :- In Decision Trees if we get real valued values in a feature we sorted them and tried all values to determine our Threshold ζ but here what we do is we try out a random sample of possible value of let's say 10 values to determine ζ



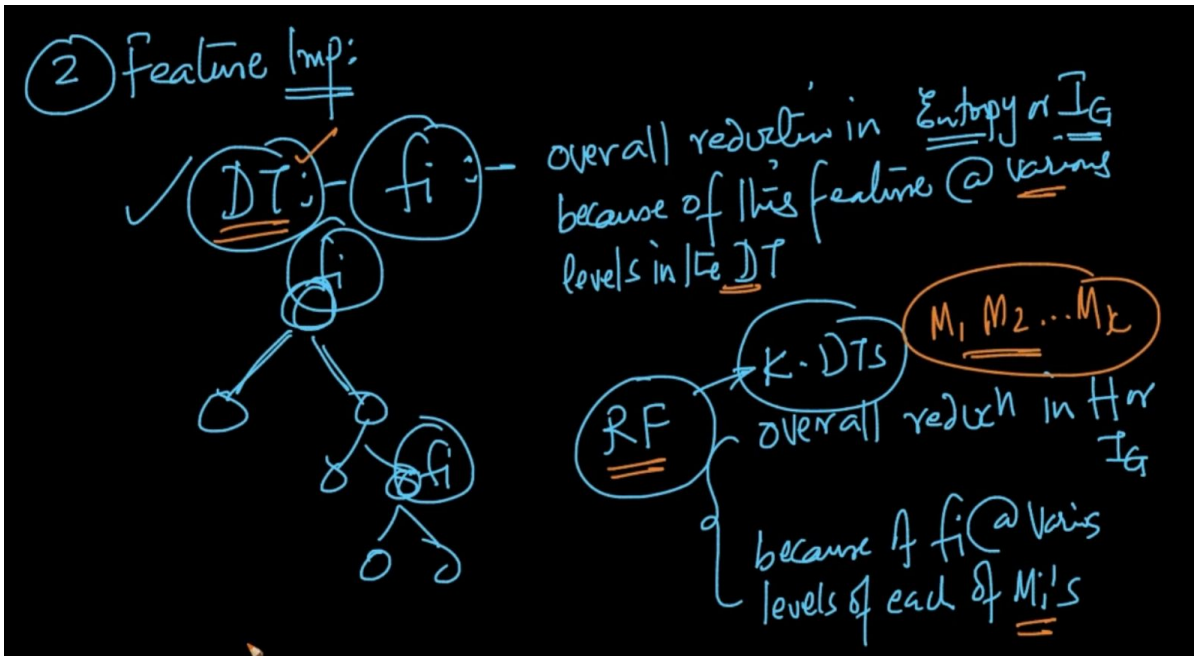
In Extreme Trees along with all the things of Random Forests we add one extra layer of Randomization when we select ζ randomly. We reduced Randomization to reduce Variance so in Extreme Trees with added layer of Randomization it reduces Variance better than Random Forest

CASES IN RANDOM FORESTS



As Random Forests are made with Decision Trees the cases are same as Decision Trees for Random Forests except 2 cases

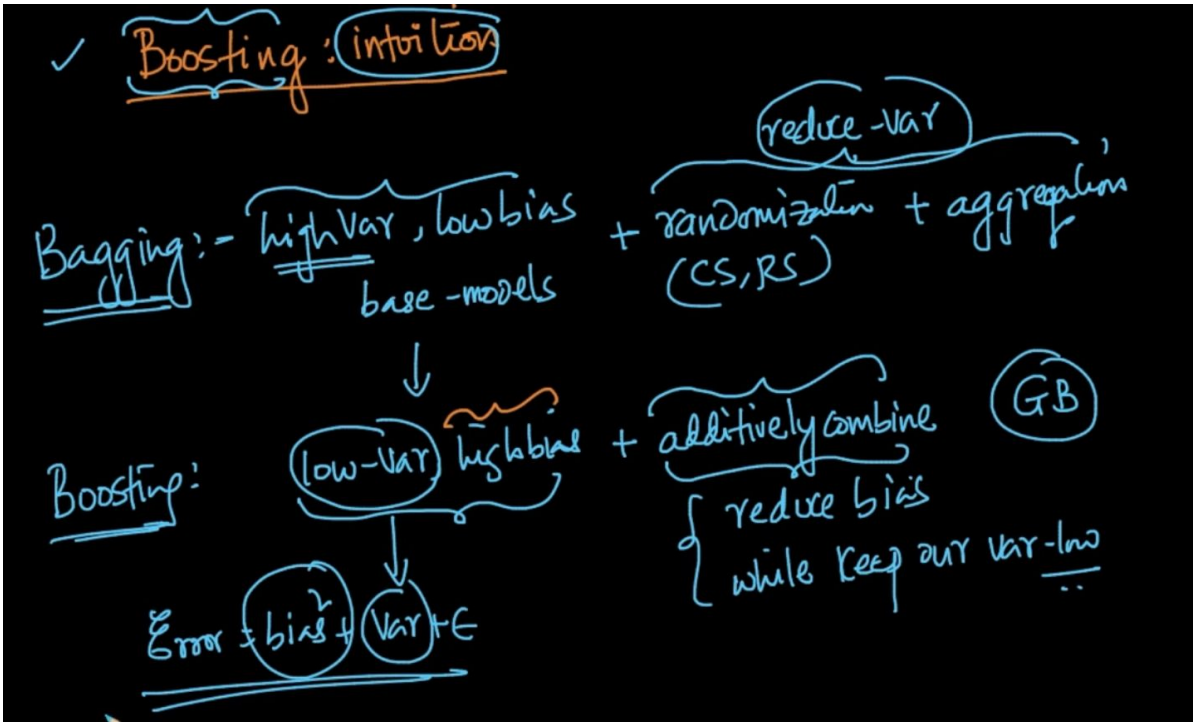
1) Bias-Variance of Random Forest : Here we use deep-trees so to reduce variance we use K - No. of Base Learners to reduce variance whereas in Decision Trees for handling Bias Variance we use the depth of Trees



2) Feature Importance : In Decision trees we calculated reduction in Entropy or I_G because of a feature at various levels in for deciding feature in DT

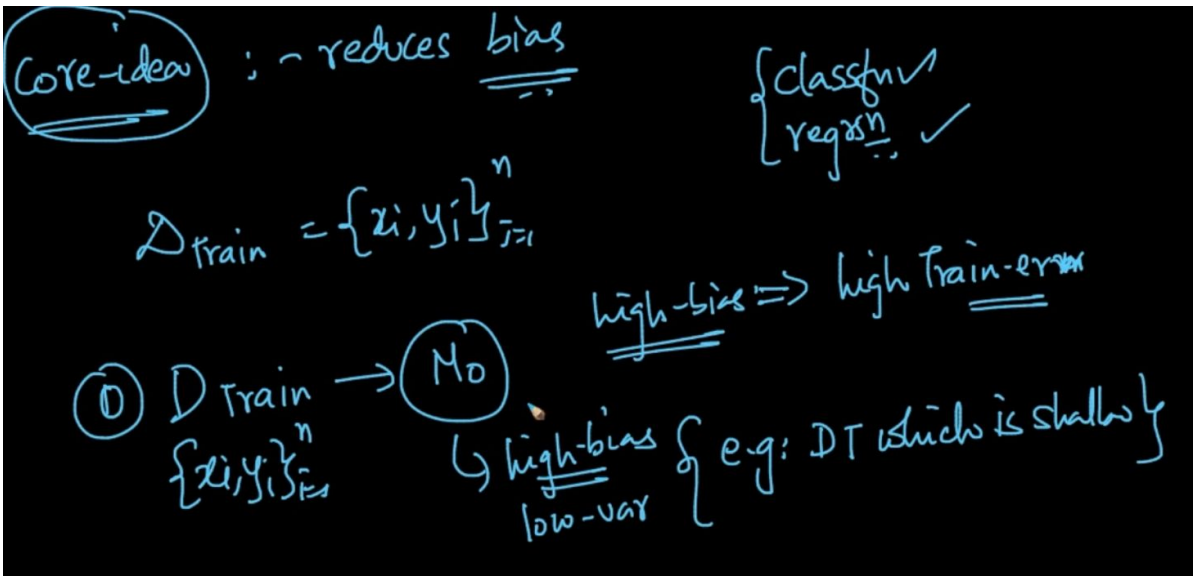
In Random Forest , we calculate overall reduction in Entropy or I_G because of a feature at various levels of each base Learner/ Model $M_i's$ as we deal with multiple DT here

PART 2 : BOOSTING INTUITION



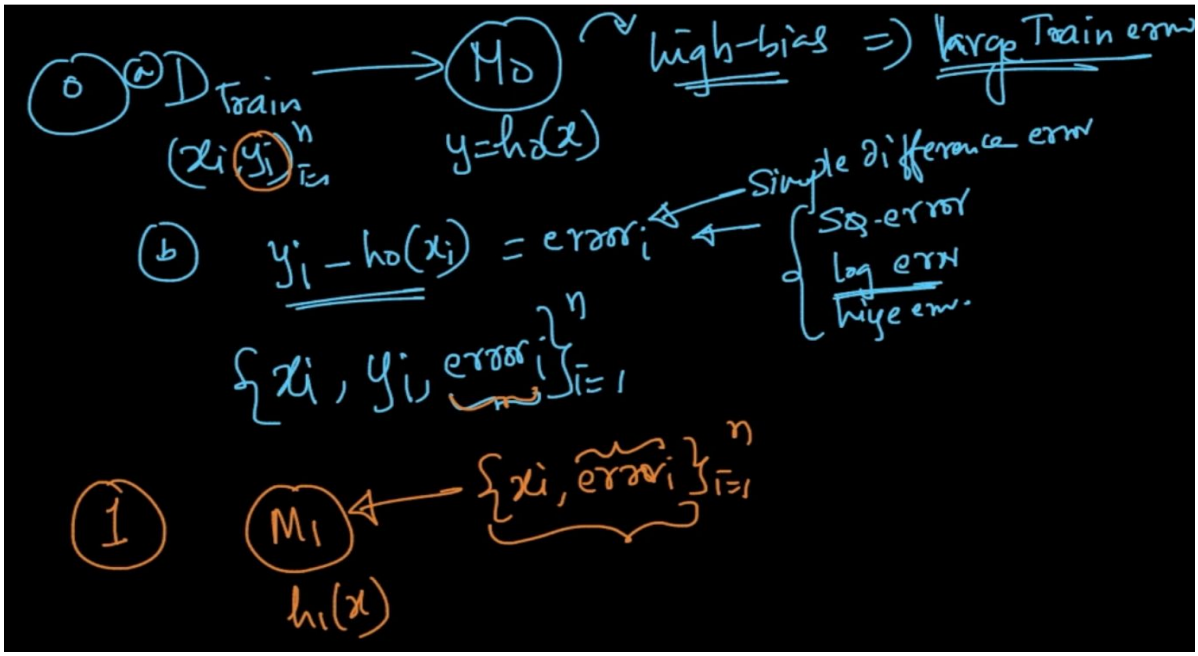
Bagging : We did randomization (Column Sampling,Row sampling) + Aggregation to reduce variance

Boosting : We have low variance and high Bias and to reduce bias we additively combine to do it to keep variance low



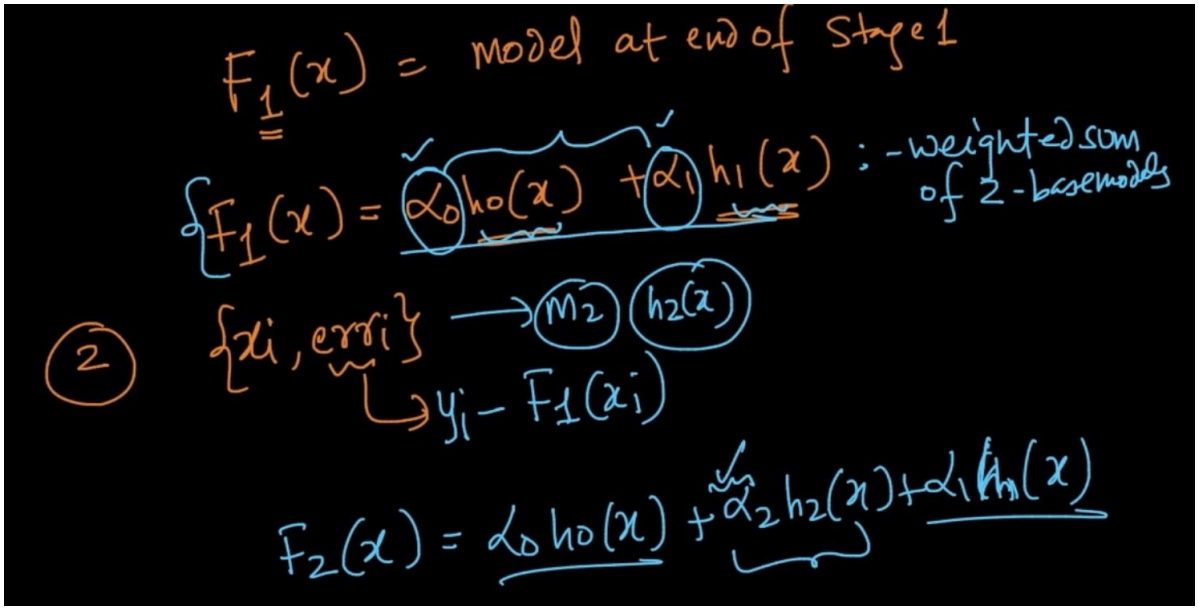
Our core idea is to reduce Bias. So let's take down our steps

Step 0 :- a) We've Train Data D_{Train} we train a model M_0 with all the D_{Train} and the model has high bias and low variance (Ex: A shallow DT (low depth)). High bias means high Train error.



Step 0 :- b) After making model M_0 we have y = $h_0(x)$. Since it's a high bias model it has large Train error. We calculate $error_i$ = $y_i - h_o(x_i)$ and store it

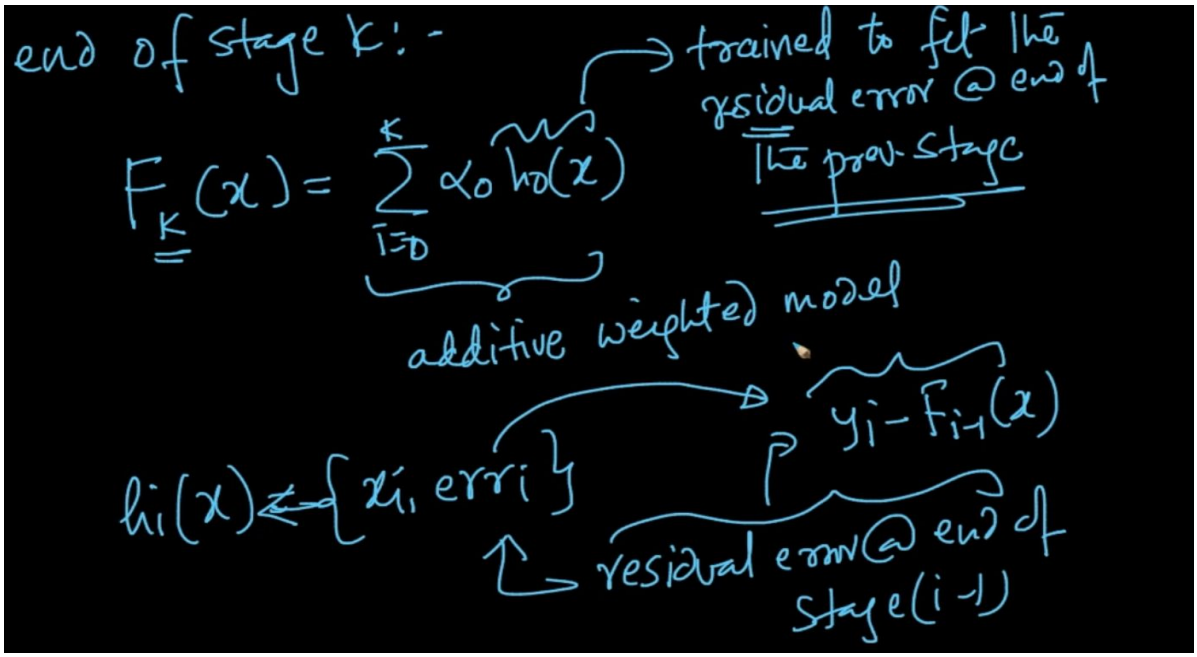
Step 1 :- Now we create another model M_1 with $\{x_i, error_i\}_{i=1}^n$ where we use $error_i$ as y instead of y_i and we'll get $h_1(x)$



At the end of Step 1 we'll have model $F_1(x) = \alpha_0 h_0(x) + \alpha_1 h_1(x)$. We can see it as a weighted sum of 2-base models .

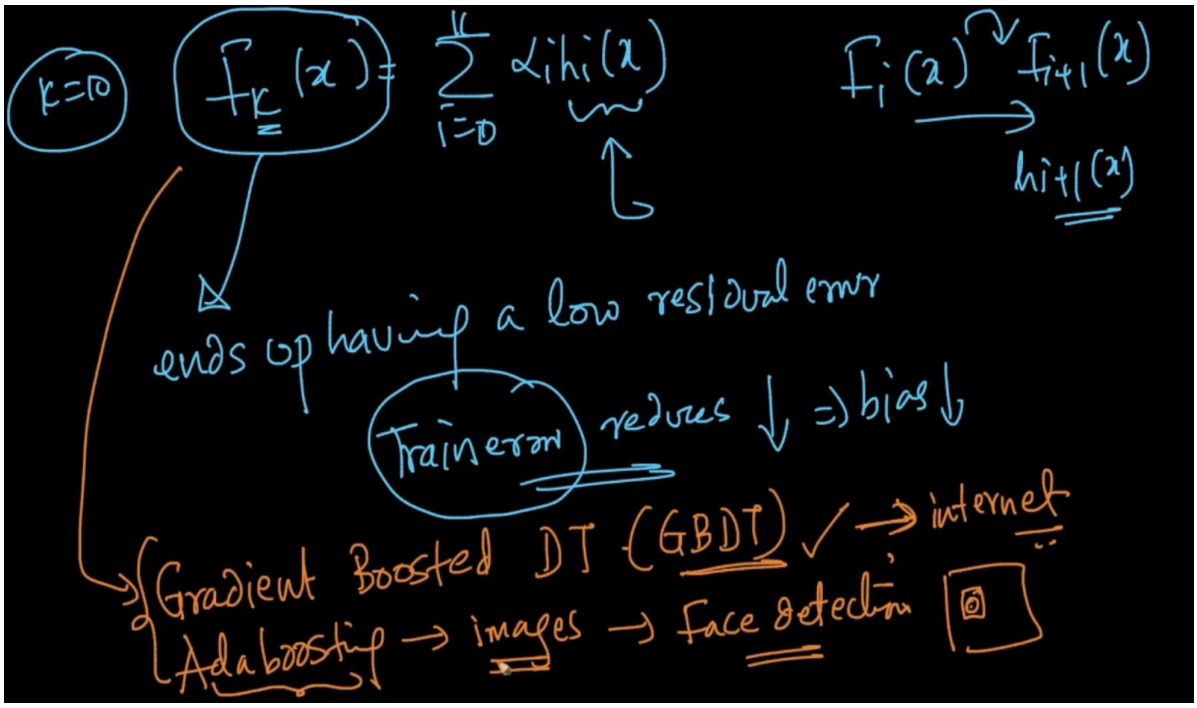
Note : α's are just coefficients . We'll learn how to calculate them

Step 2 : We create a model M_2 with $\{x_i, error_i\}_{i=1}^{n}$ where $error_i = y_i - F_1(x)$ and we'll get $h_2(x)$. Now we do $F_2(x) = \alpha_0 h_0(x) + \alpha_1 h_1(x) + \alpha_2 h_2(x)$ and we'll repeat this process



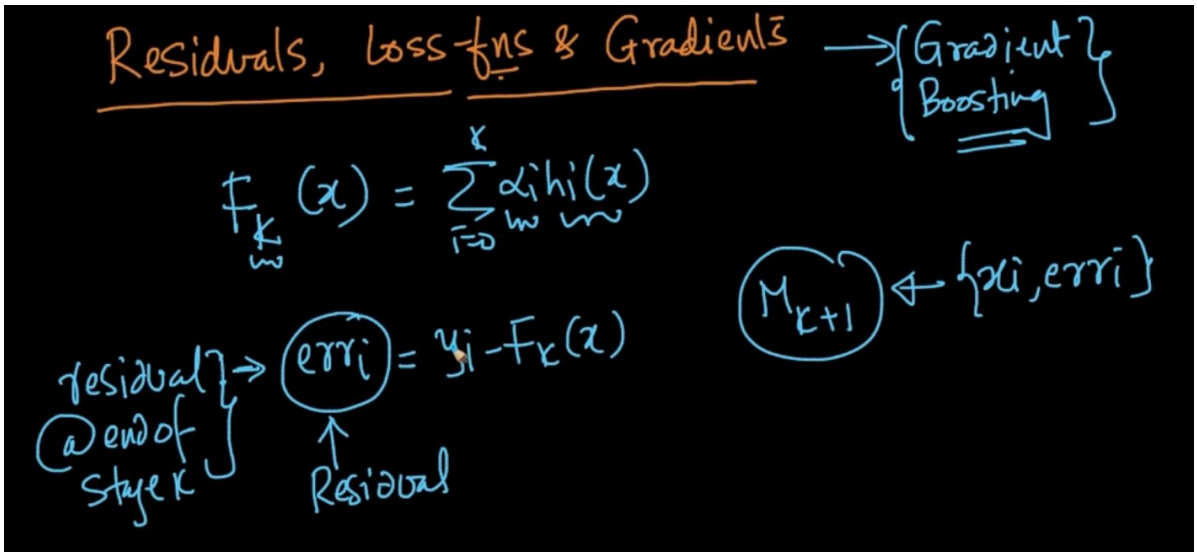
So at the end of k-steps we'll have $F_k(x) = \sum_{i=0}^{k} \alpha_i h_i(x)$ where h(x) is trained to fit residual error at the end of previous stage as $h_i(x)$ is trained on $\{x_i, error_i\}$ and error is residual or left error at stage (i-1) or the previous stage => Mathematically writing: $y_i - F_{i-1}(x)$

Note : "k" is a hyper-parameter

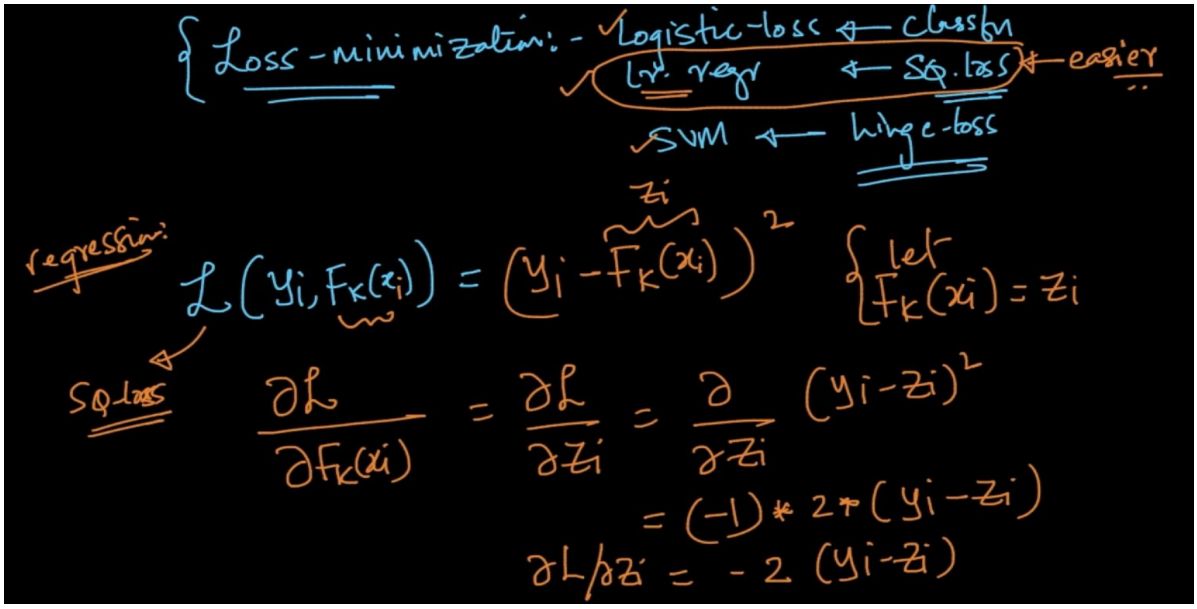


Our $F_k(x) = \sum_{i=0}^{k} \alpha_i h_i(x)$ is basically summation of models and we end up with a low residual error model because our Training error is reduced. Why's it reduced ? As we go from $F_i(x)$ to $F_{i+1}(x)$ our Train error gets reduced because we are fitting $h_{i+1}(x)$ with the previous error so it's getting reduced and as it reduced our bias also gets reduced

RESIDUALS, LOG-LOSS AND GRADIENTS



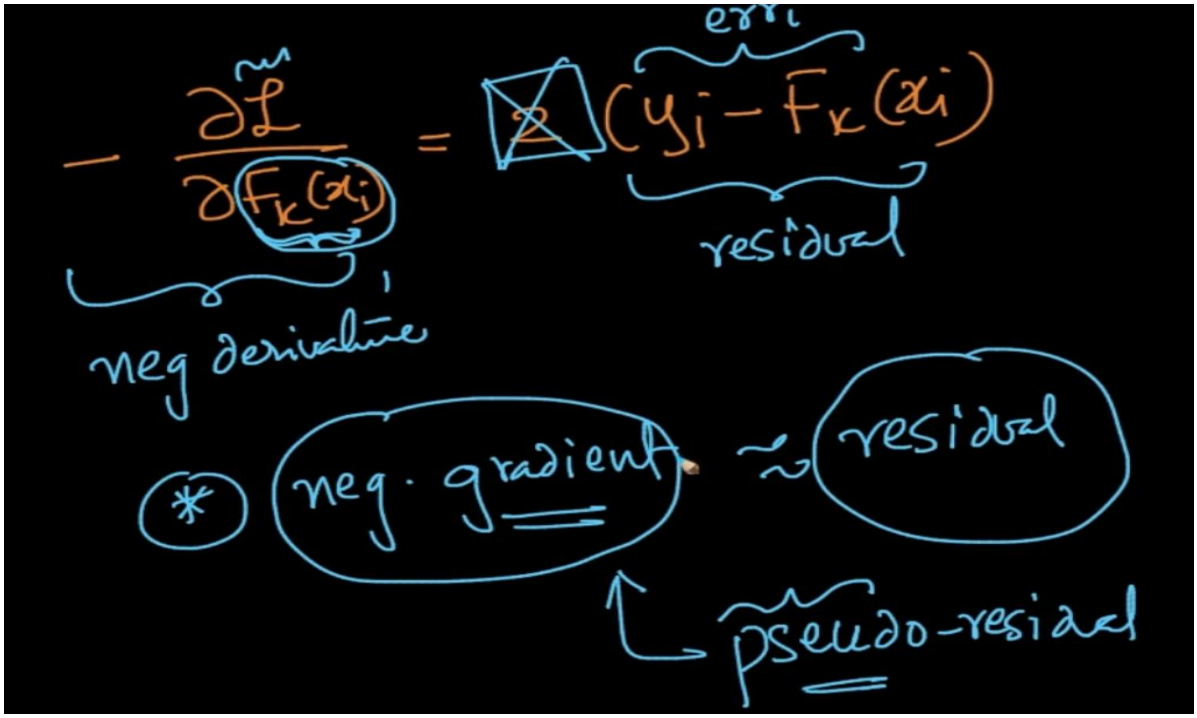
Residual at end of stage 'k' => $error_i = y_i - F_k(x)$ and a Model M_{k+1} is trained with $\{x_i, err_i\}$



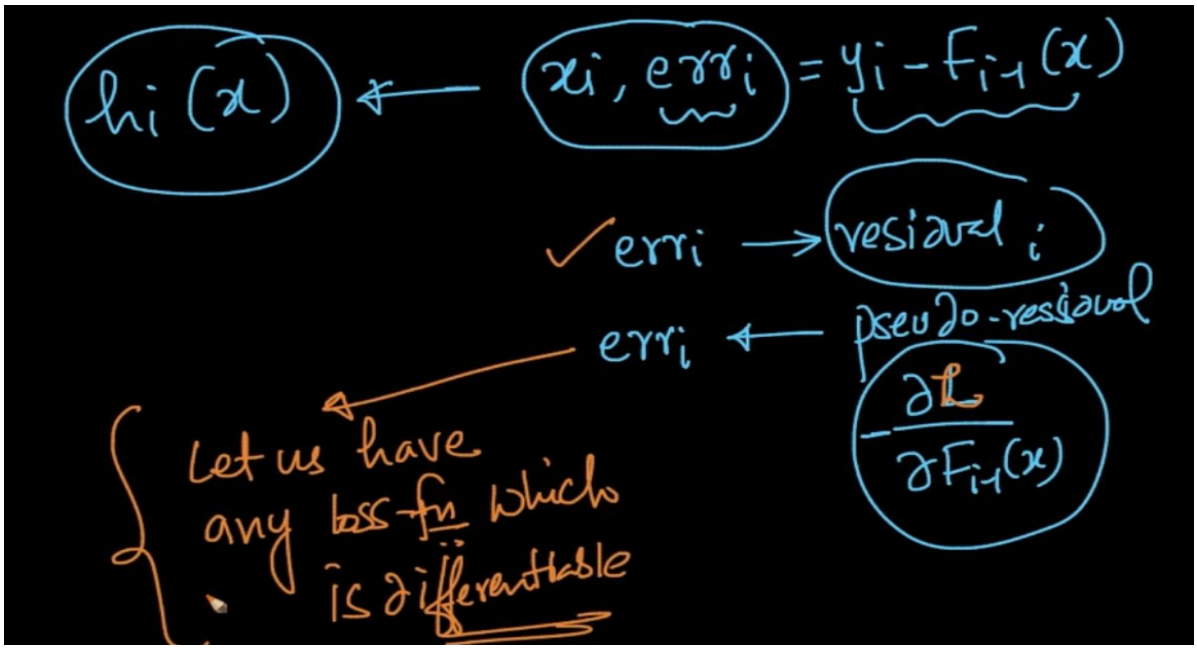
In ML, we try do Loss minimization as mentioned above for the algos like SVM, Logistic and Linear Regression. SO if we take example of Linear Regression our loss function

$L(y_i, F_k(x)) = (y_i - F_k(x_i))^2$ let $F_k(x) = z_i$

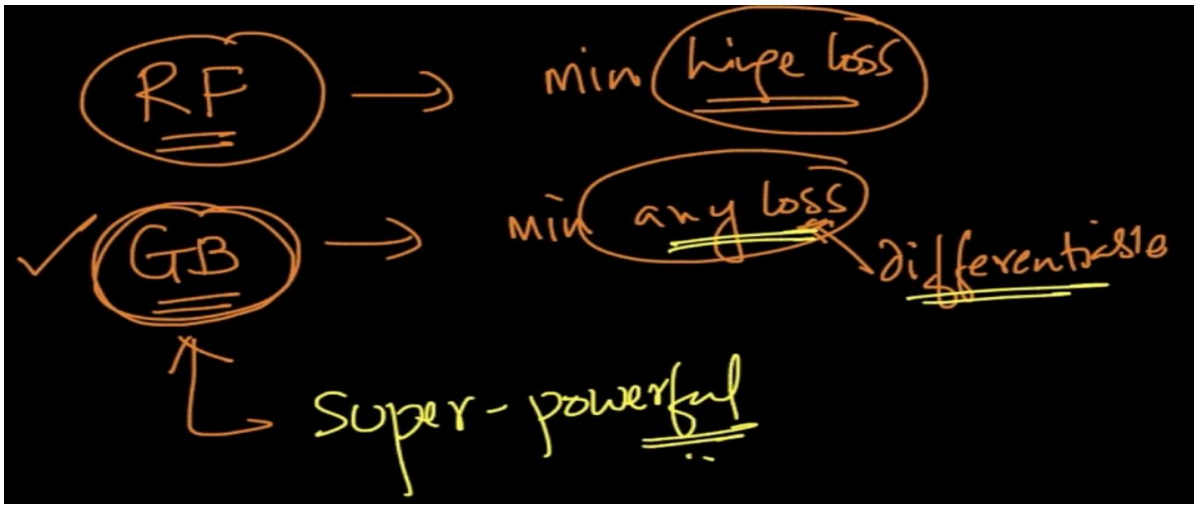
So $\frac{\delta L}{\delta z_i} = \frac{\delta}{\delta z_i}(y_i - z_i)^2 = -2 * (y_i - z_i)$



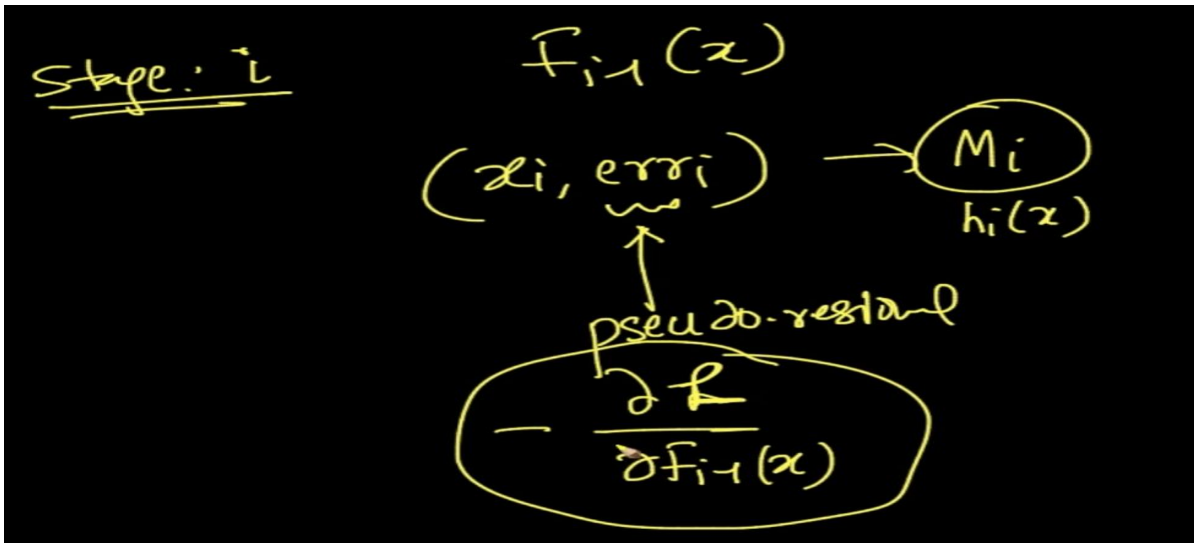
- $\frac{\delta L}{\delta F_k(x_i)}$ = 2 $(y_i - F_k(x_i)$ so we can say that our negative gradient / derivative i.e differentiation of L w.r.t our function F of x at k stage i.e $F_k(x) \approx$ residual. Our negative grad can also be called pseudo-residual



Our $h_i(x)$ is trained with x_i, err_i where err_i -> residual. Instead of residual we can have pseudo residual i.e - $\frac{\delta L}{\delta F_{i-1}(x_i)}$. The advantage of pseudo residual is that it can let us have any loss function which is differentiable since our pseudo residual is $\left(- \frac{\delta L}{\delta F_{i-1}(x_i)}\right)$

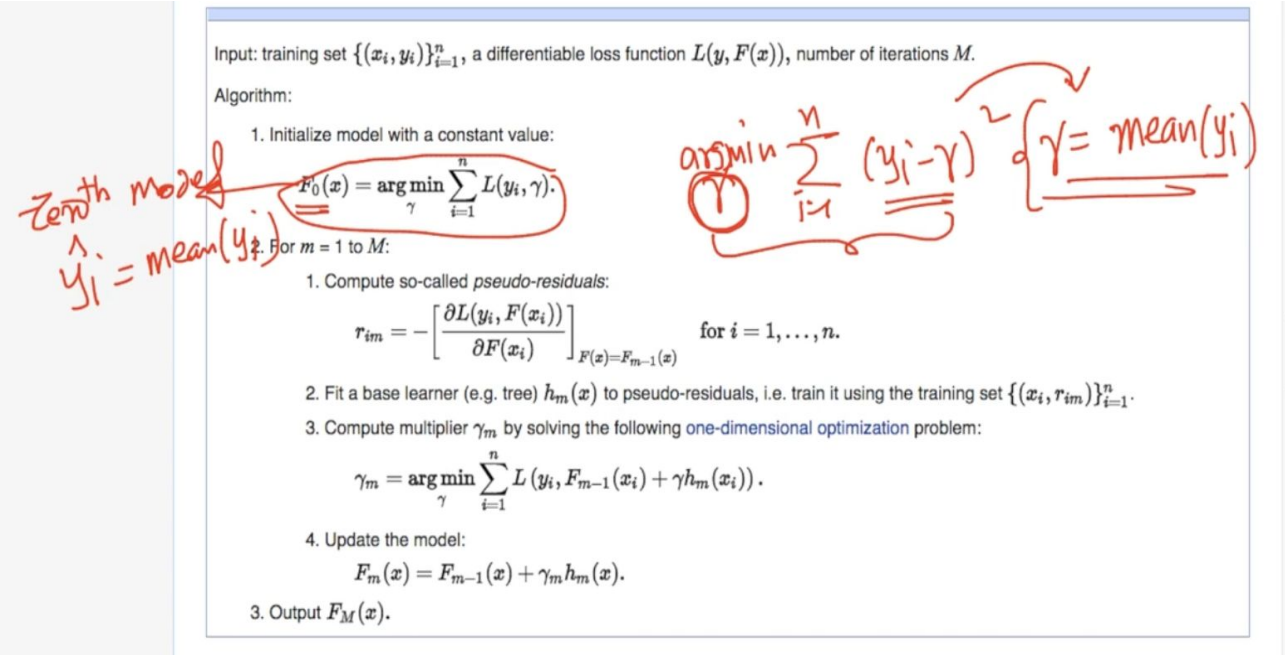


In Random forest we couldn't minimize hinge loss but with Gradient Boosting we can minimize any loss which is differentiable. This makes Gradient Boosting extremely powerful



At stage i : We already have $F_{i-1}(x)$ we train our model M_i i.e $h_i(x)$ with $(x_i, error_i)$ where error is pseudo residual => $\left(- \frac{\delta L}{\delta F_{i-1}(x_i)}\right)$

GRADIENT BOOSTING



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 constant value:

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

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 (e.g. tree) $h_m(x)$ 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:

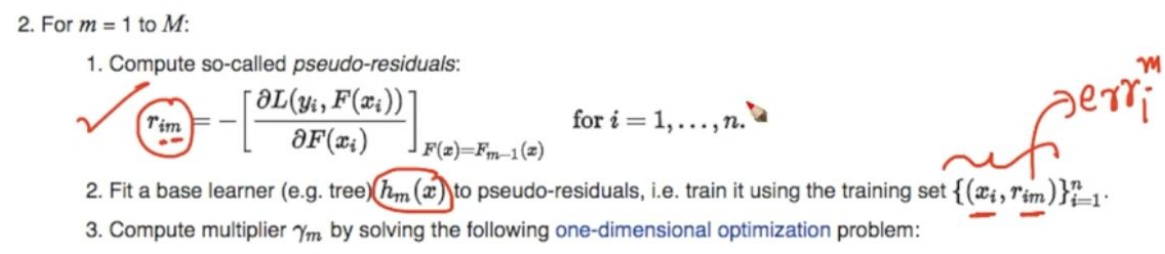
$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

   4. Update the model:

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

3. Output $F_M(x)$.

For initializing, let's say we've a Squared loss. We need to find γ such that it minimizes the loss function. In squared loss the γ = mean(y_i). So that's how we are initializing our base/Zeroth model



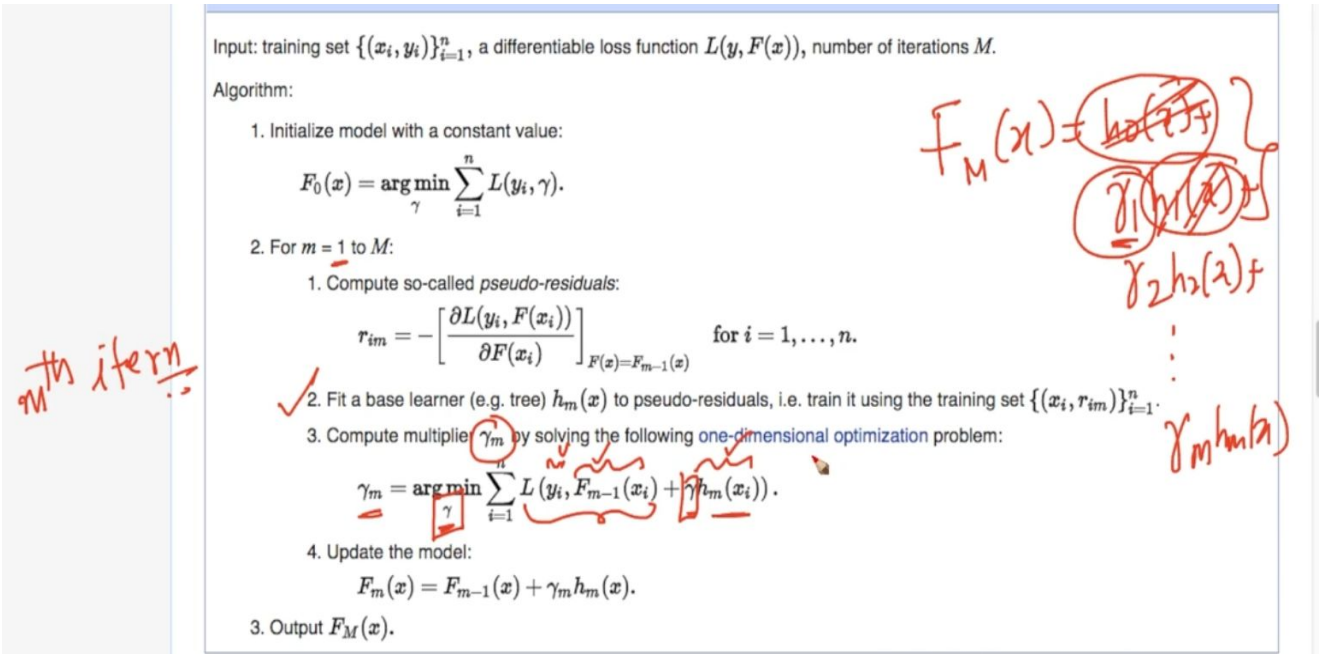
2. For $m = 1$ to M:

   1. Compute so-called *pseudo-residuals*:

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

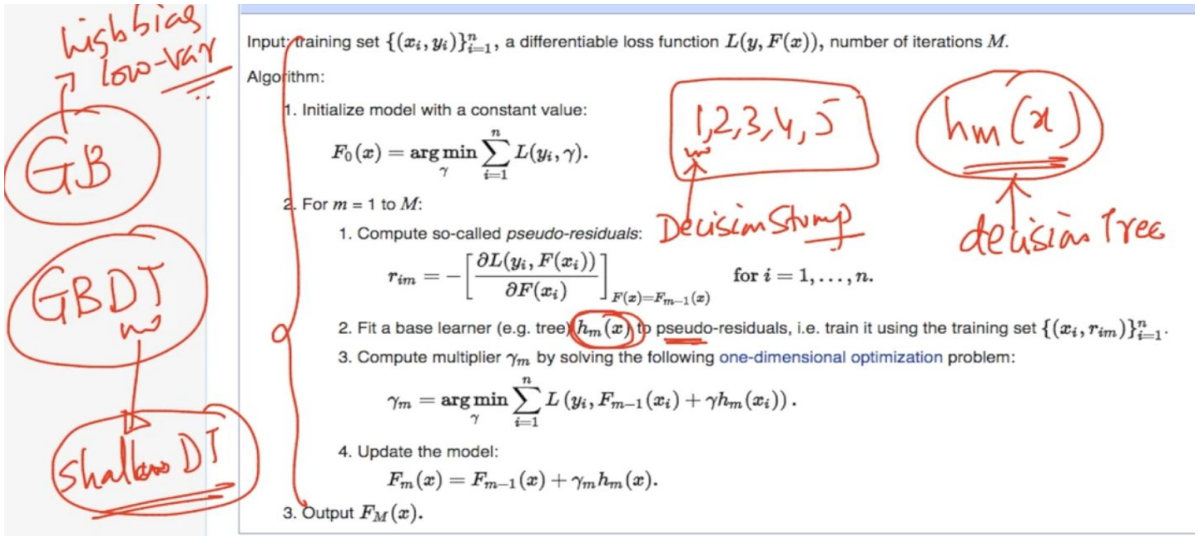
   2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^n$.
   3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

Now we calculate the Pseudo-residuals as mentioned. Our train data is (x_i, r_{im})



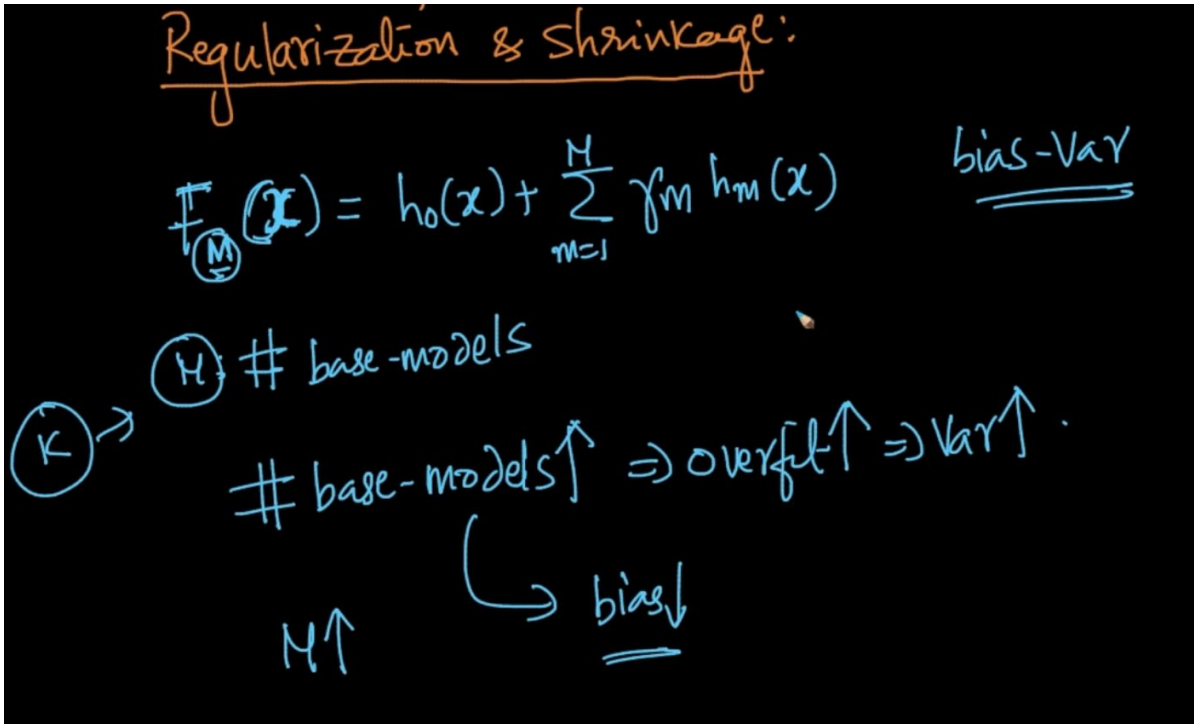
$F_m(x) = h_0(x) + \gamma_1 h_1(x) + \gamma_2 h_2(x) \ldots\ldots \gamma_m h_m(x)$

We calculate y_m at m stage = $min \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i)) + \gamma h_m(x_i))$ we need to find γ which minimizes this whole thing. Now we update our model. So our $F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$ Boosting is an additive technique.

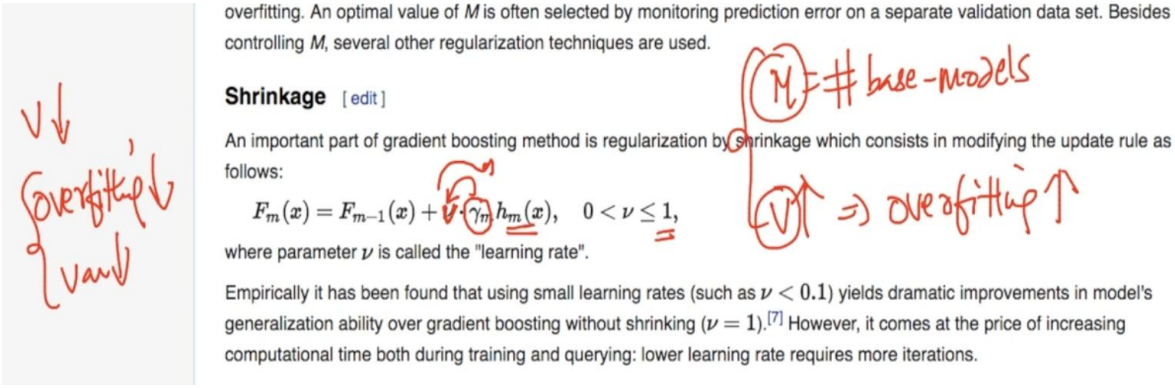


Our base learners $h_m(x)$ is a decision tree . In Gradient Boosting we need high bias and low variance so our trees are shallow here i.e of low depth

REGULARIZATION AND SHRINKAGE



M - No. of Base Models and as they increase no doubt our bias gets decreased but our overfit ↑ => variance so there's a nice idea called shrinkage to deal with this



overfitting. An optimal value of M is often selected by monitoring prediction error on a separate validation data set. Besides controlling M, several other regularization techniques are used.

Shrinkage [edit]

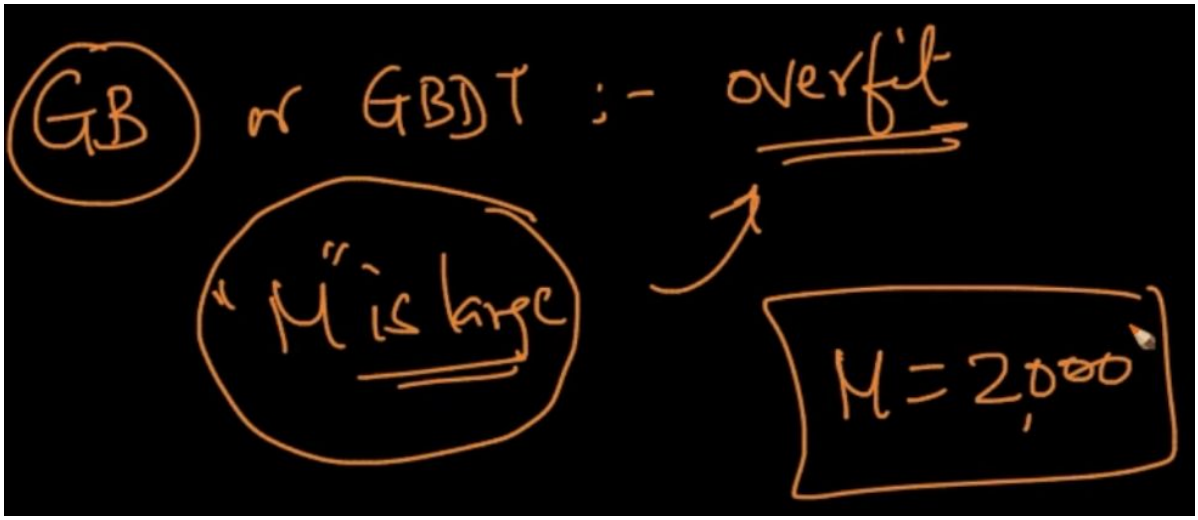
An important part of gradient boosting method is regularization by shrinkage which consists in modifying the update rule as follows:

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

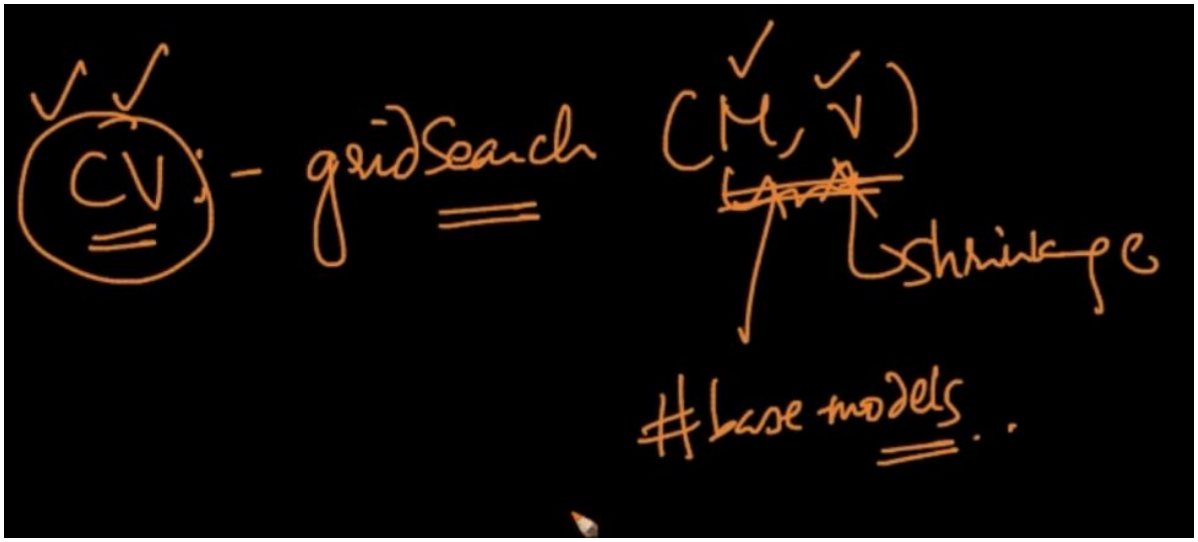
where parameter ν is called the "learning rate".

Empirically it has been found that using small learning rates (such as $\nu < 0.1$) yields dramatic improvements in model's generalization ability over gradient boosting without shrinking ($\nu = 1$).[7] However, it comes at the price of increasing computational time both during training and querying: lower learning rate requires more iterations.

$F_m(x) = F_{m-1}(x) + \nu * \gamma_m h_m(x)$ so we add ν called as "learning rate". It is also a parameter As we can see if it's small then it reduces the weight of $\gamma_m h_m(x)$ so our chances of overfitting also decreases. So we can imply that as $\nu \downarrow$ => overfitting , variance $\downarrow$ and as $\nu \uparrow$ => overfitting $\uparrow$. In Gradient Boosting M and ν are hyper-parameters

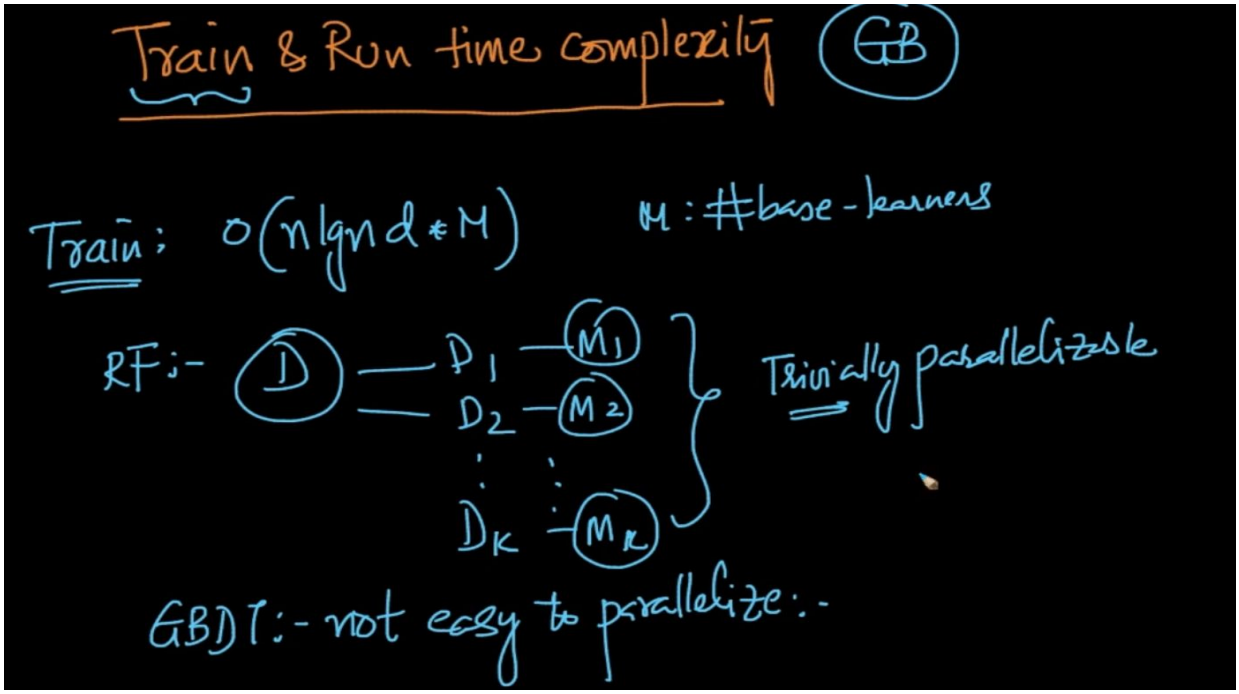


It’s very common in Gradient Boosting that people overfit by taking large M

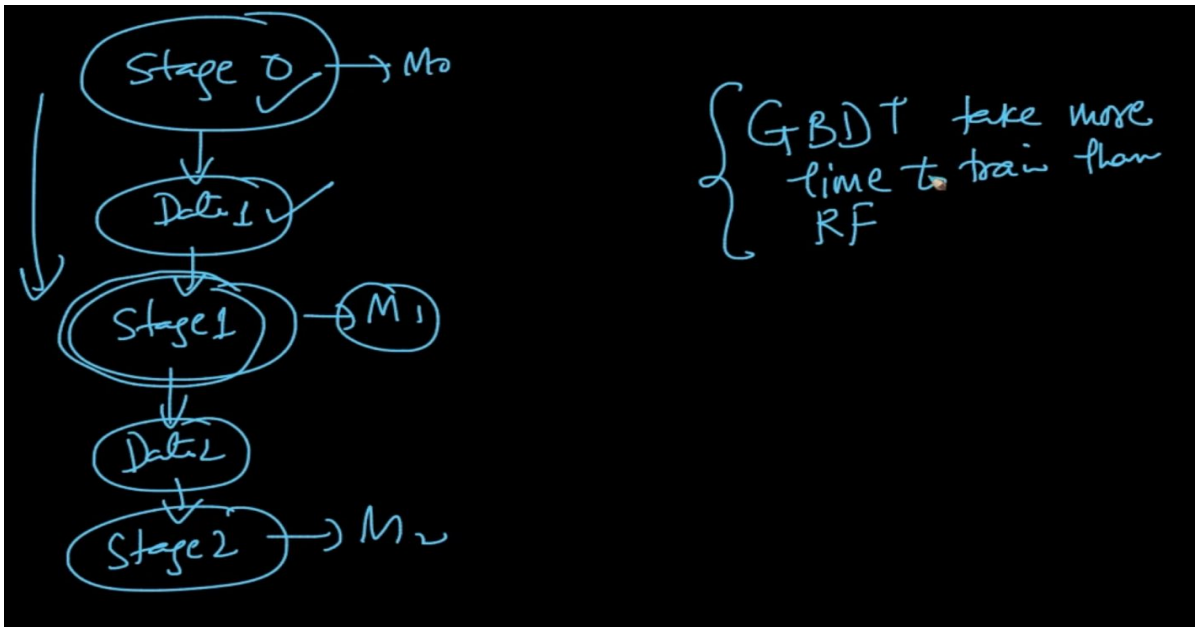


Always do Cross Validation i.e GridSearch to get the best M and ν. Not only here but in every model

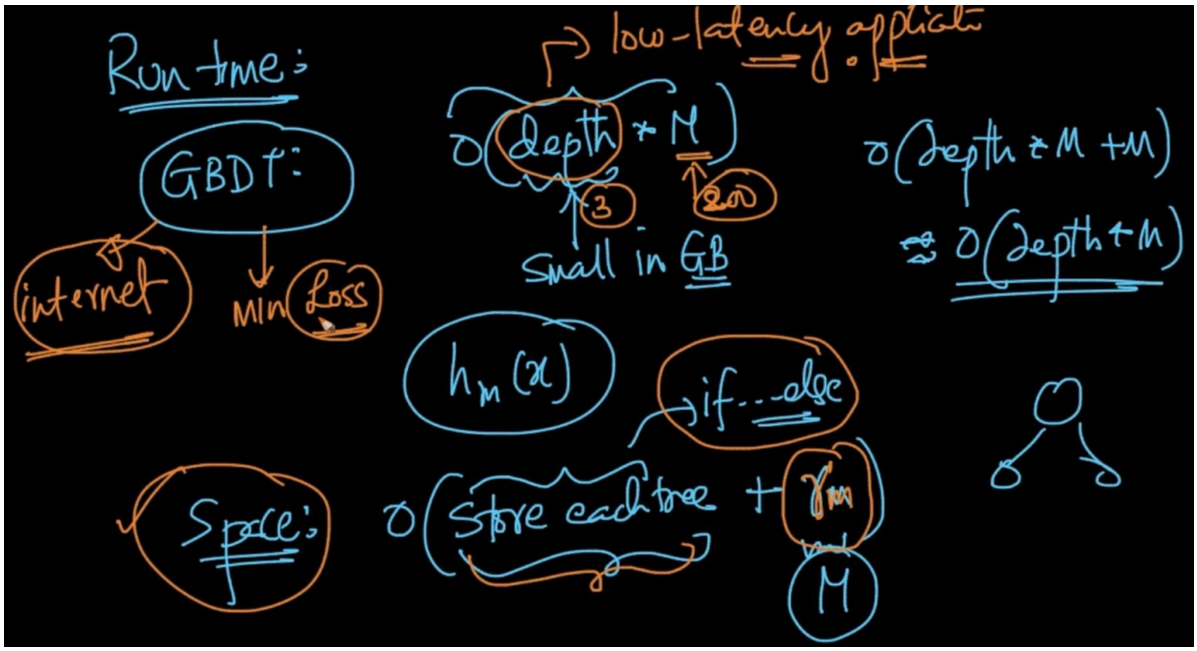
TRAIN AND RUN TIME COMPLEXITY



Train time complexity : O (nlogn *d * M) where O(n * logn) is Time complexity of DT and M is no. of base learners. Random Forest was parallelizable as each models were separate from other model but GBDT is not easy to parallelize as it's a very sequential model



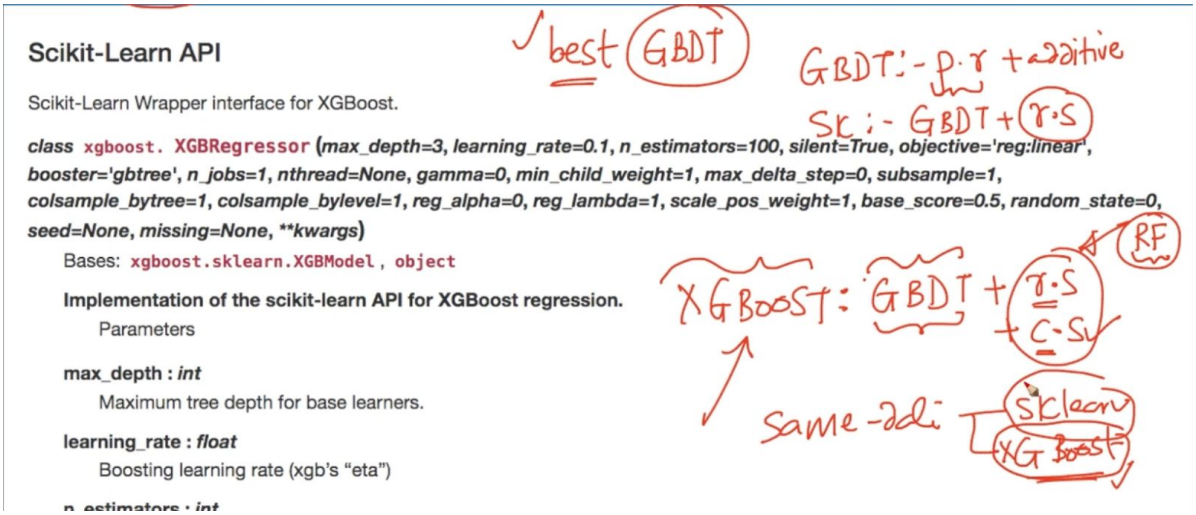
As we can see in Gradient Boosting we train model M_0 at stage 0 and from the data obtained we train model M_1 and so on . Since it's not parallelizable and sequential it takes more time than Random Forest



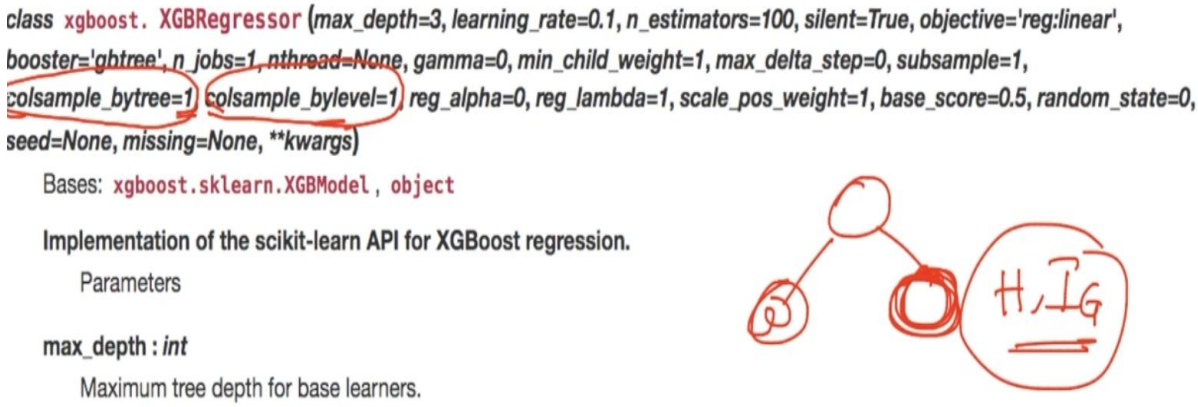
Run Time Complexity: O(depth * M) but one of the interesting thing is that depth is small in GB so it makes extremely fast and can be used in low latency applications

Space Complexity : O(Store each tree + γ_M) and storing each tree is just like Random Forest but we also need to store the γ_M for each base learner

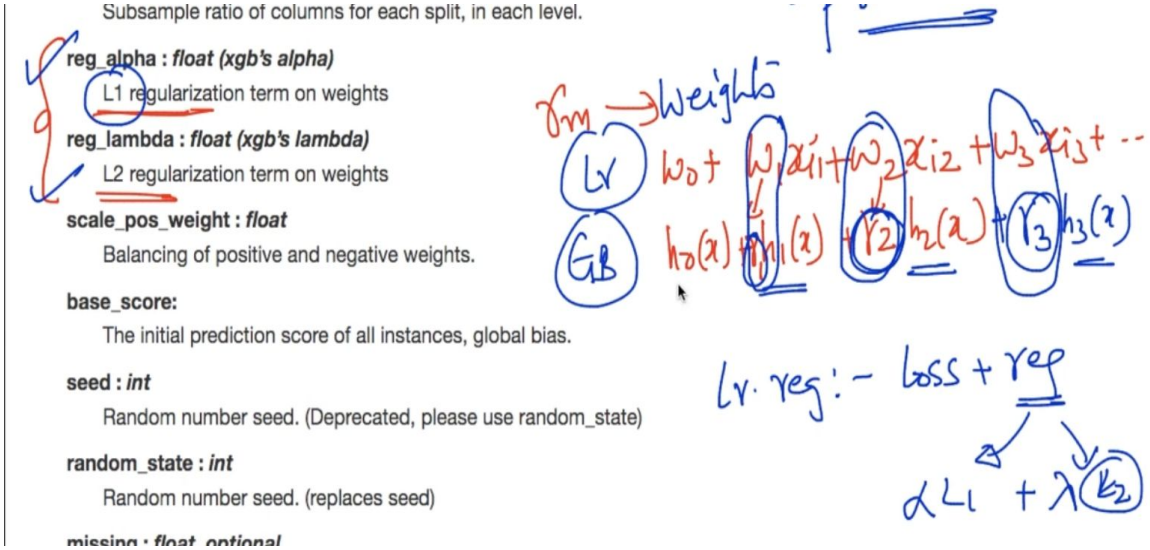
XGBOOST



XGBoost : Here it takes all the great things mentioned from Gradient Boosting + It also takes great things of Random Forests like Row Sampling and Column Sampling



The most important thing in XGB is that you can sample columns by trees i.e we don't need all features for making a tree. We can just use some random columns as well

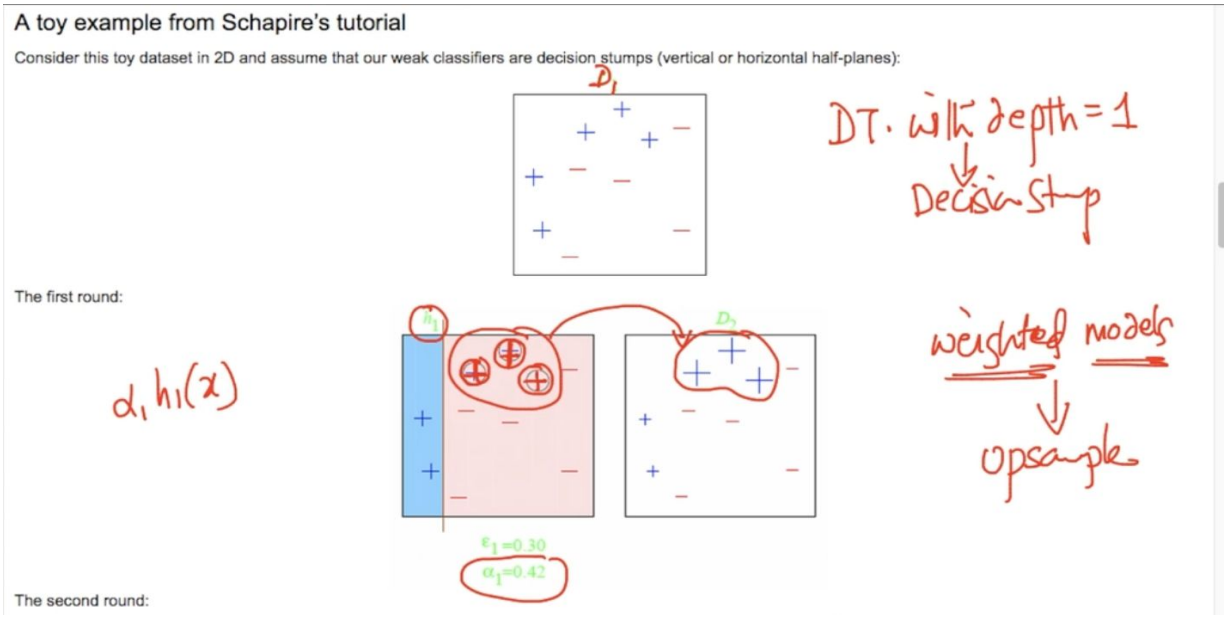


In Linear Regression : $w_0 + w_1 x_1 + w_2 x_2 + ...$

In Gradient Boosting : $h_0(x) + \gamma_1 h_1(x) + \gamma_1 h_1(x) .....$

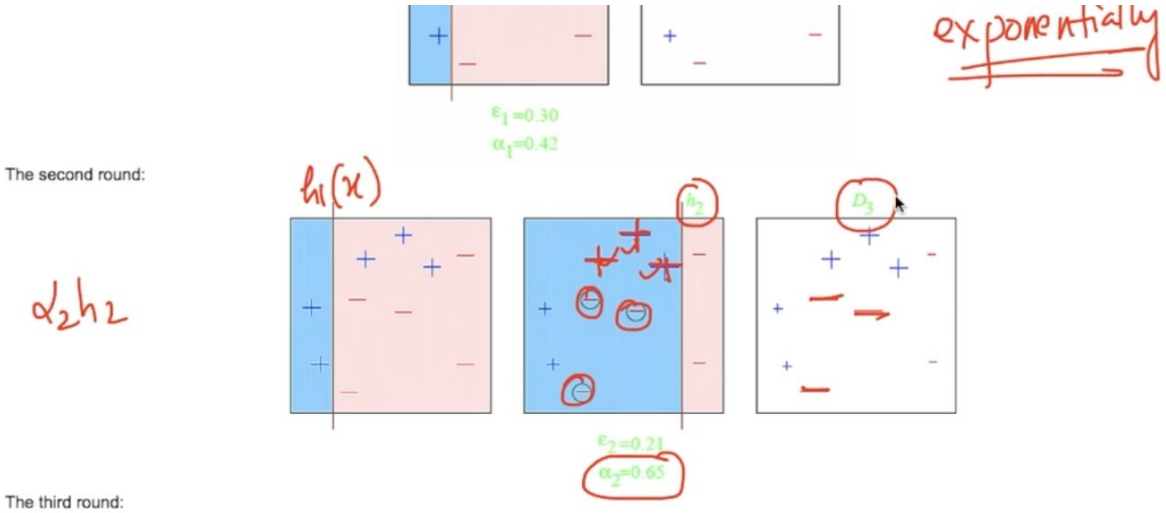
We did L1/ L2 regularization on weights . We can do that here as well since we can consider our $\gamma's$ as weights and perform Regularization

ADABOOST



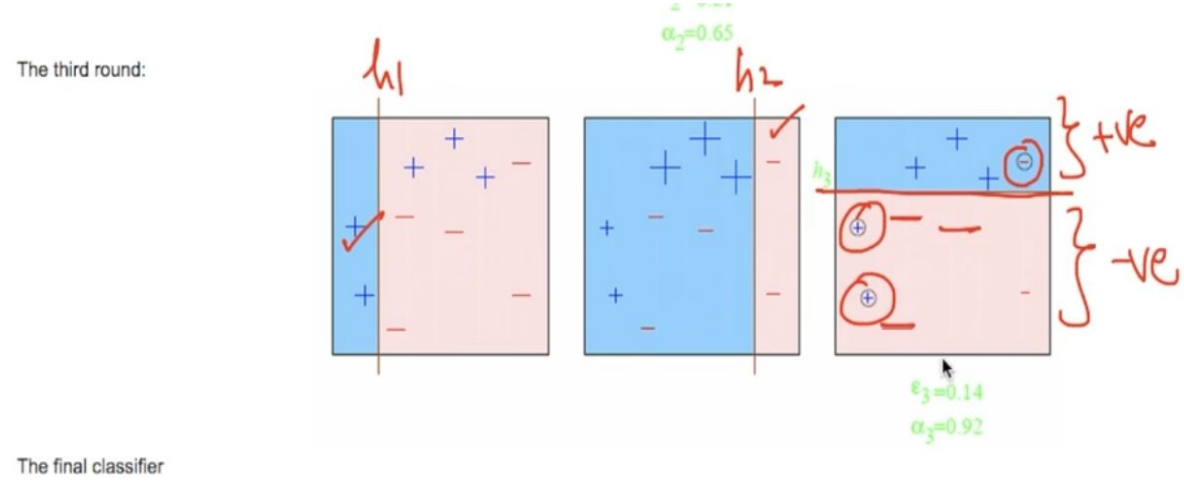
We've dataset D1 as shown we train a DT with depth = 1 i.e with 1 hyperplane

When we create our first model $h_1(x)$ we also get weight α based on errors.We get 3 erroneous points as shown above. We give it more weight i.e upsample and that will be used as a dataset in 2nd model

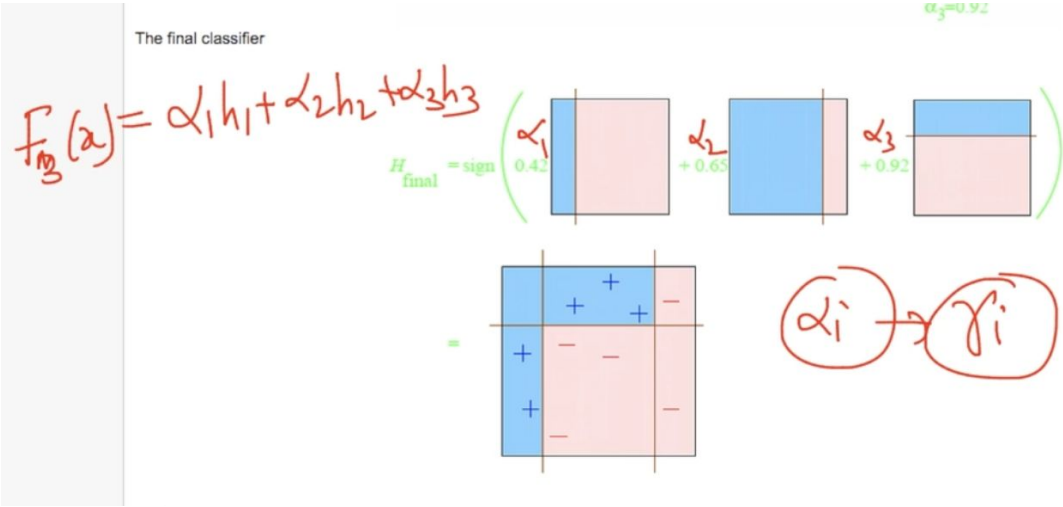


Now we create a model h_2 separating the upsampled points correctly. We get α_2 as well

Now also 3 error pts. We'll upsample it



Now we train our model h_3 . It has 2 error points but it's taken care by h_1

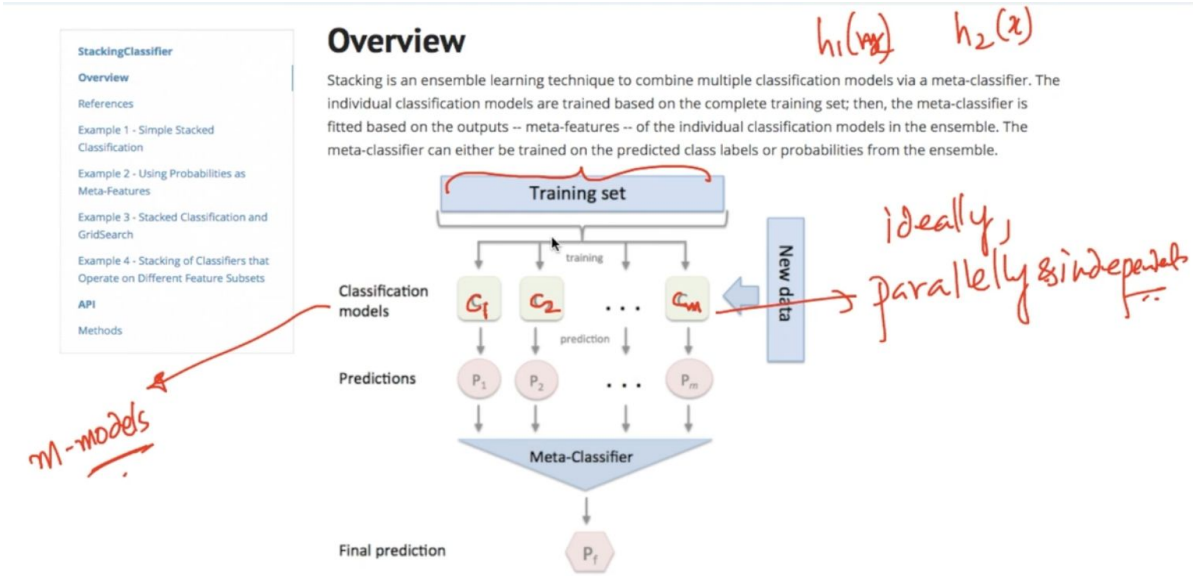


So our final model is made by combining all the 3 models. Our final model

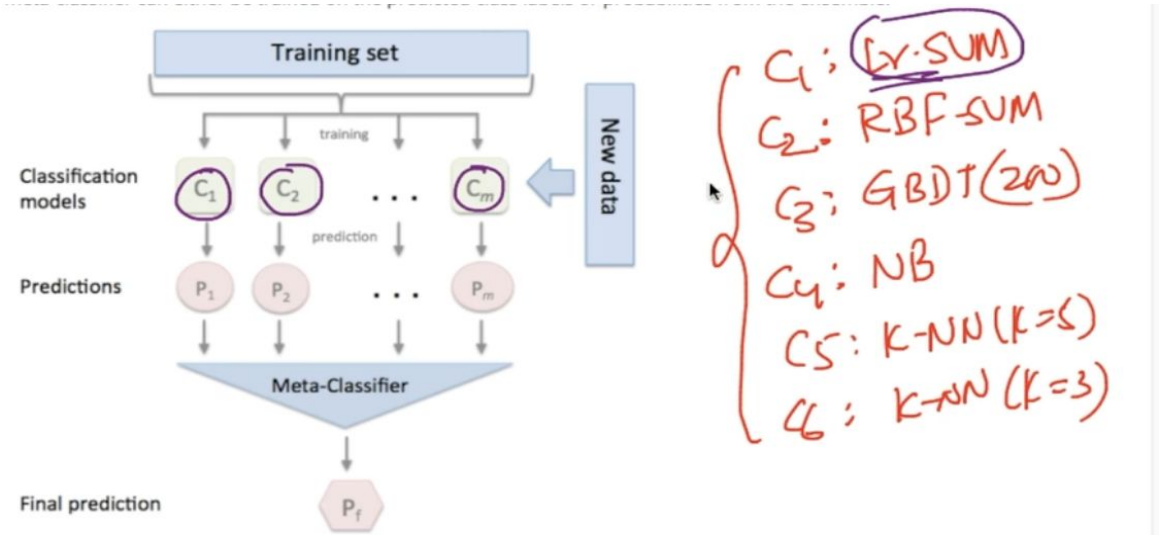
$F_3(x) = \alpha_1 h_1 + \alpha_2 h_2 + \alpha_3 h_3$

In AdaBoost, our model is adapting to the errors of the previous model and more importance is given to the points which are misclassified.This is core of Adaboost

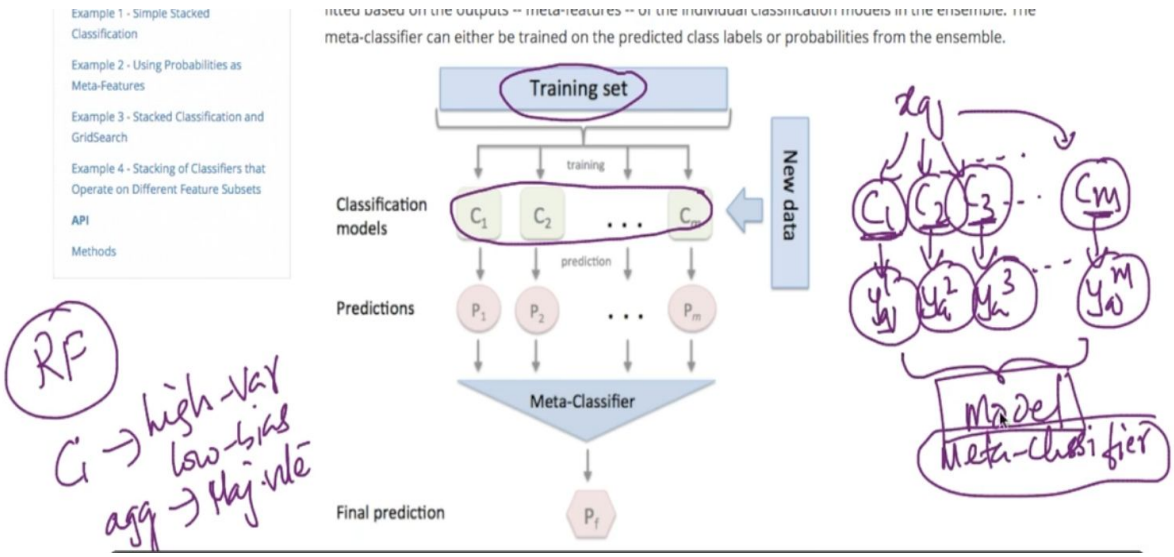
STACKING MODELS



We train m-models and each model is parallel and independent from each other



As we can see each model is different from another one and the more different it is the better output



So we have x_q and we train m-models i.e $C_1,\ C_2,\ C_3, \ldots.\ C_m$ and we get some y_q from each of them i.e $y_q^1,\ y_q^2,\ y_q^3, \ldots y_q^m,$ we combine them and put it into a model called meta-classifier. Remember in Random Forests we trained multiple models that had high variance and low bias .We combined them and used majority vote to reduce variance but here all the base models C are well-tuned with good bias-variance tradeoff and our Meta-Classifier is also an ML model like Logistic Regression or any other instead of aggregation function like majority vote

The algorithm can be summarized as follows (source: [1]):

Algorithm 19.7 Stacking

Input: Training data $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^{m}$ ($\mathbf{x}_i \in \mathbb{R}^n$, $y_i \in \mathcal{Y}$)
Output: An ensemble classifier H

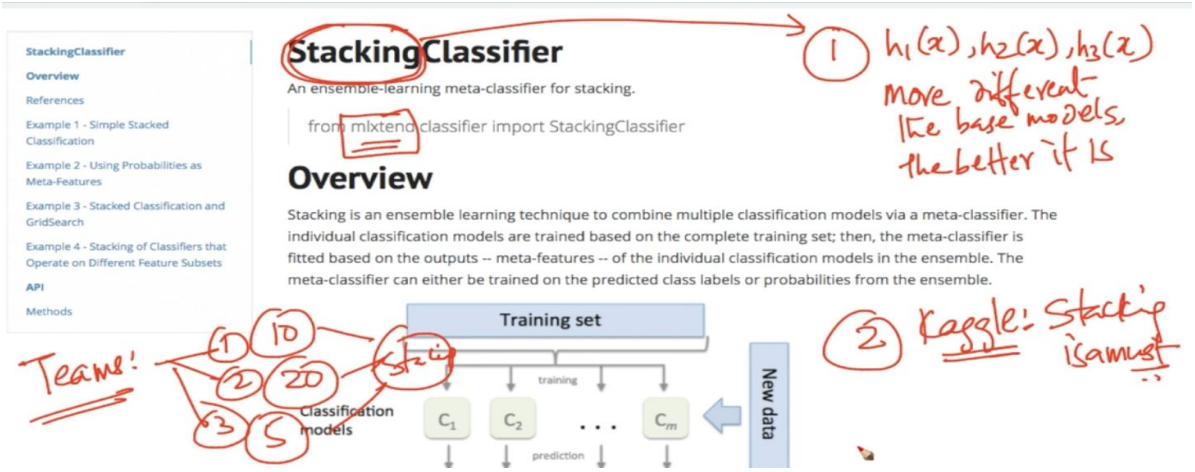
1: Step 1: Learn first-level classifiers
2: **for** $t \leftarrow 1$ to T **do**
3: Learn a base classifier h_t based on $\mathcal{D}$
4: **end for**
5: Step 2: Construct new data sets from $\mathcal{D}$
6: **for** $i \leftarrow 1$ to m **do**
7: Construct a new data set that contains $\{\mathbf{x}'_i, y_i\}$, where $\mathbf{x}'_i = \{h_1(\mathbf{x}_i), h_2(\mathbf{x}_i), \ldots, h_T(\mathbf{x}_i)\}$
8: **end for**
9: Step 3: Learn a second-level classifier
10: Learn a new classifier h' based on the newly constructed data set
11: **return** $H(\mathbf{x}) = h'(h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_T(\mathbf{x}))$

Step 1: We have dataset D as shown we train T- number of classifiers h and each classifiers have proper bias variance tradeoff .

Step 2 : We construct new3 dataset D' where $\{x_i^{'}, y_i\}$ where $x_i^{'} = \{ h_1(x_i), h_2(x_i), \ldots.. h_T(x_i)\}$

Step 3: Now all the new Dataset D' is applied to a second-level (Meta) classifier and

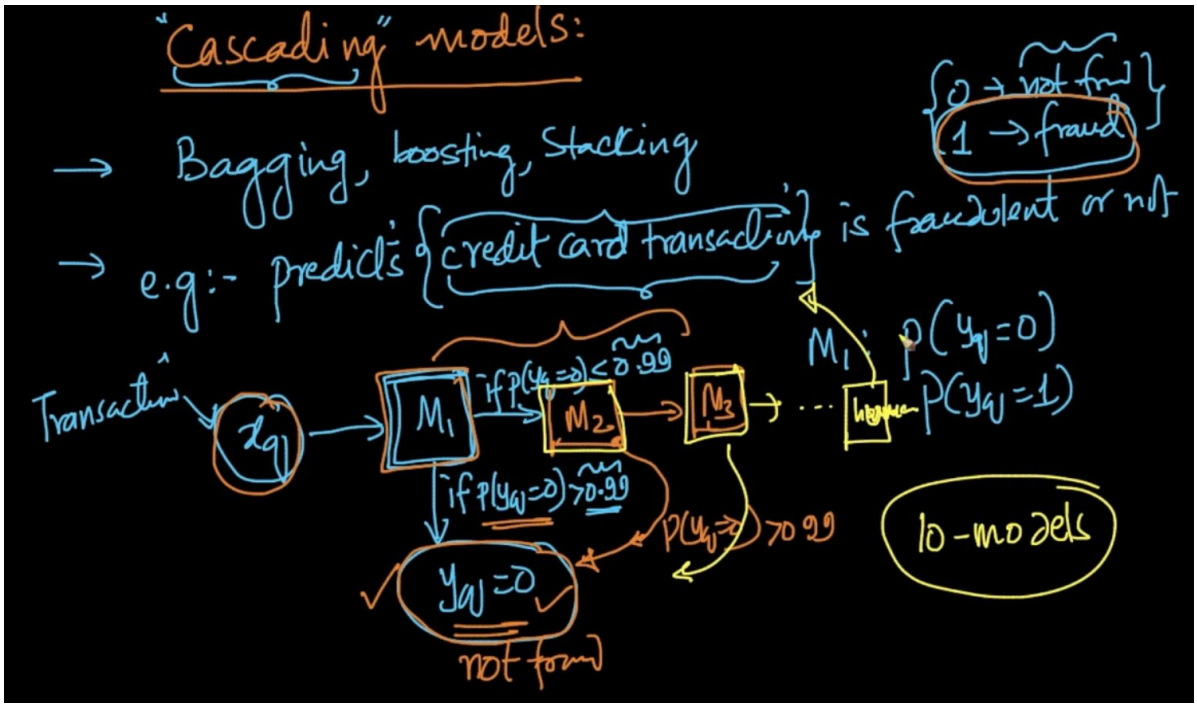
y_q = h'($h_1(x_i)$, $h_2(x_i)$,..... $h_T(x_i)$)



1) The different each base model is the better

2) Kaggle : Stacking is a must in Kaggle. In a team one can work on 10 classifiers ,other on 20 and then they can stack it but in real world it's seldom used as it has huge Train and Runtime

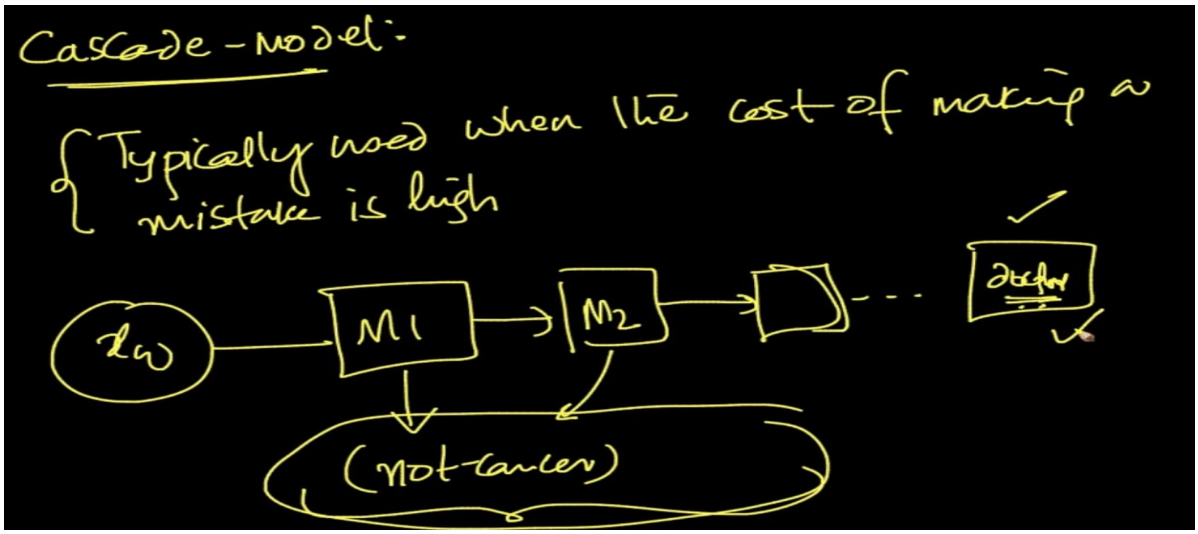
CASCADING CLASSIFIERS



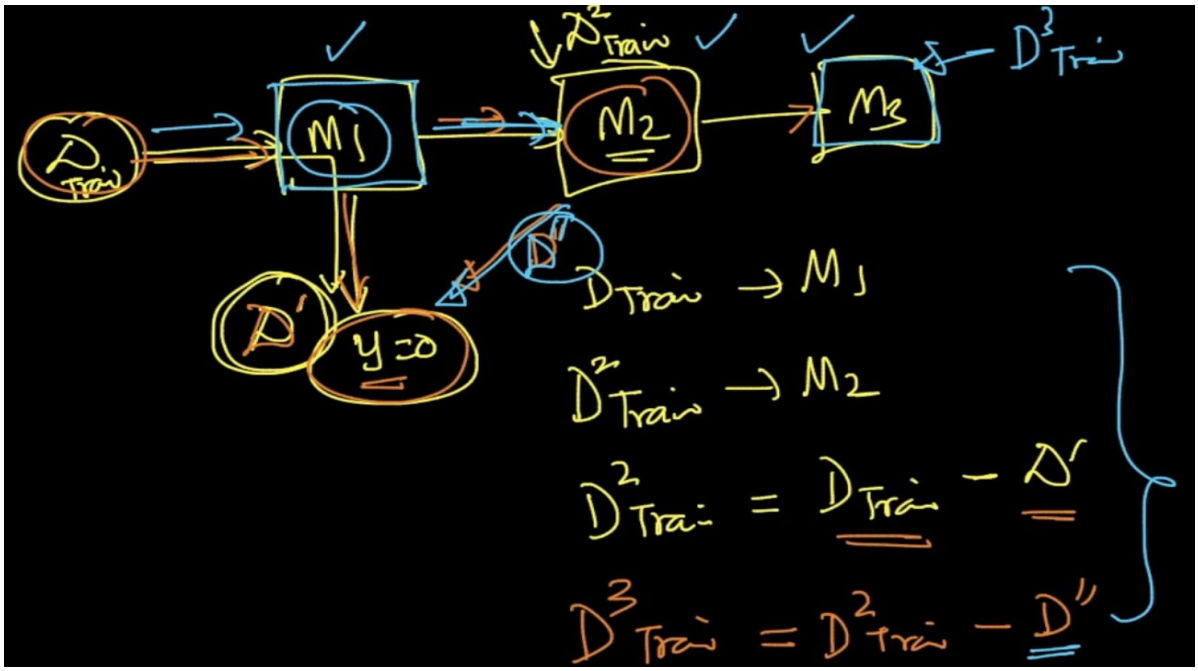
We are taking the above example of Credit card fraud detection

We've x_q we train a model M_1 with that . Our model gives P($y_q = 0$) or P($y_q = 1$)

So after M_1 if P($y_q = 0$) > 0.99 we say $y_q = 0$ (not fraud) but if P($y_q = 0$)< 0.99 we train another model M_2 and there if P($y_q = 0$) > 0.99 then y_q = 0 . We repeat this process and cascade models and after that if it comes < 0.99 then a human will determine the transaction

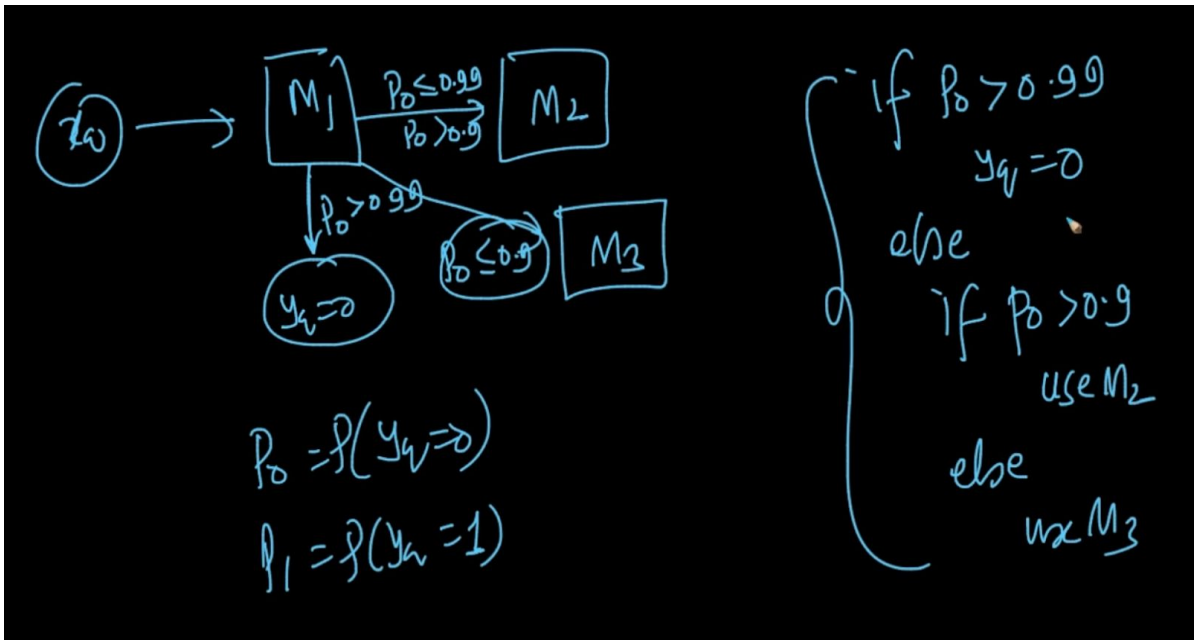


They are used when cost of making mistake is high. Ex : In Medical domain. Most of cases where algos can take care whether they are cancerous or not

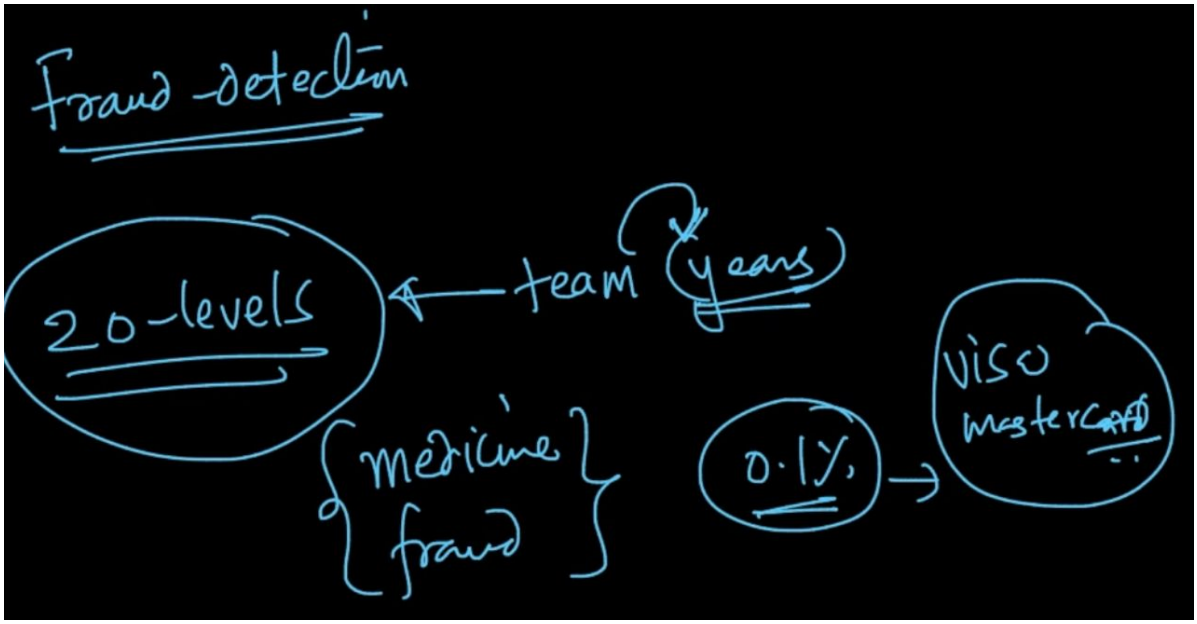


We use D_{Train} for M_1. D' points are gone into y = 0 after M_1. Now we train our model M_2 on D^2_{Train} where D^2_{Train} = D_{Train} - D' and same theory is applied for further models

We need to keep this in mind . We are not supposed to use already gone dataset again



A cascade model can also be like this . See the multiple if - else conditions above



If a team is taking years then even 20 levels of cascading blocks are added to it

Cascade models are used when cost of making mistakes is high