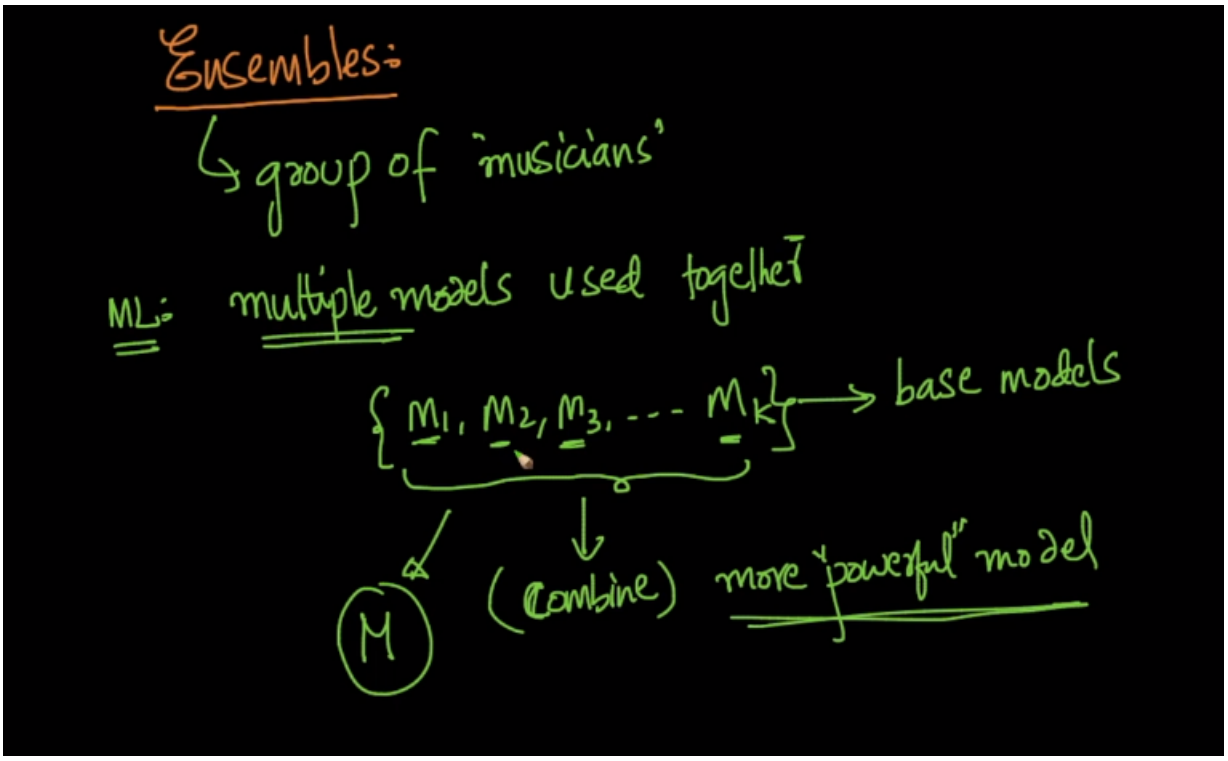
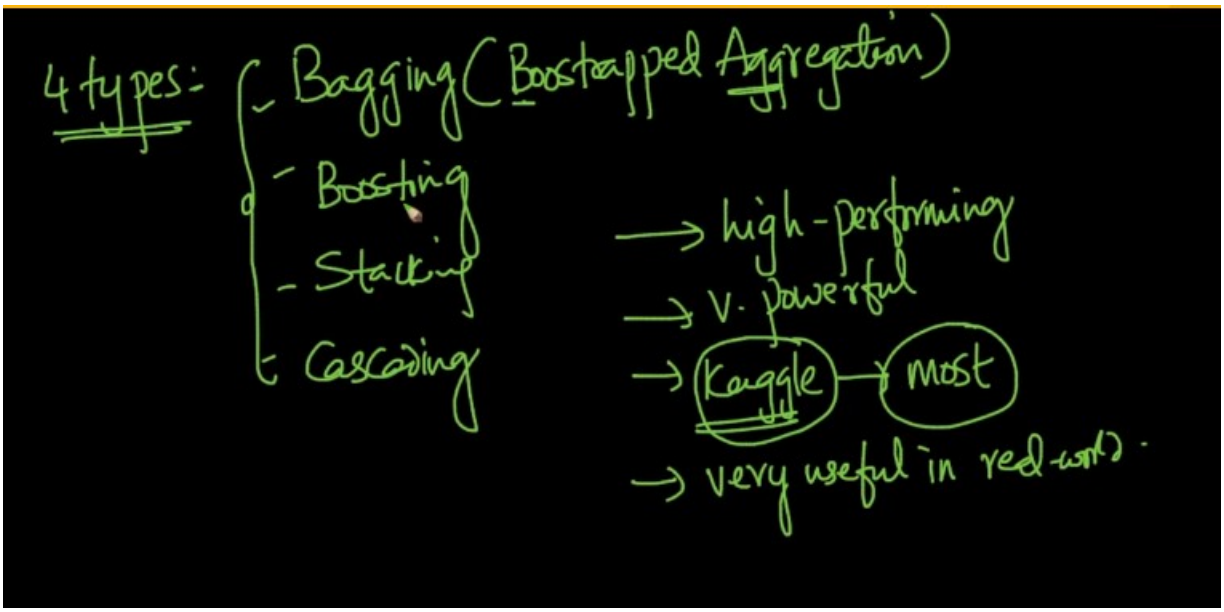
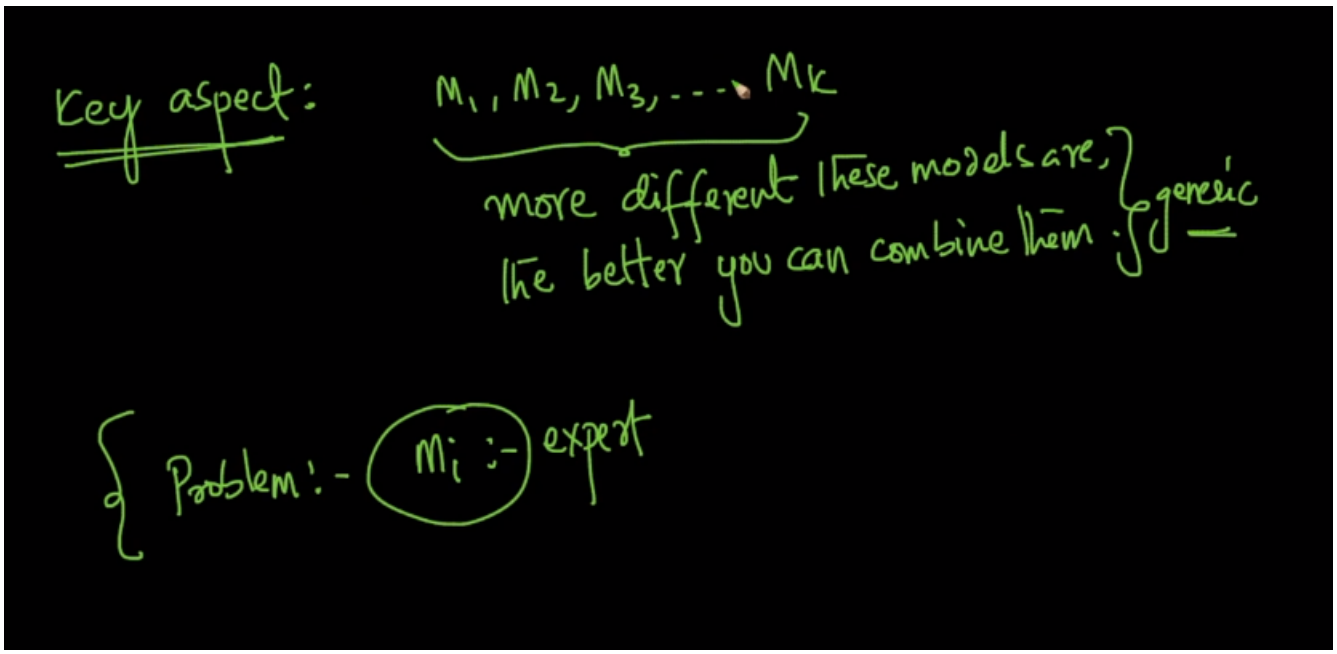
NOTES

What are ensembles?
When multiple models used together to make more powerful model. Combining the models must be done in ingeniously.



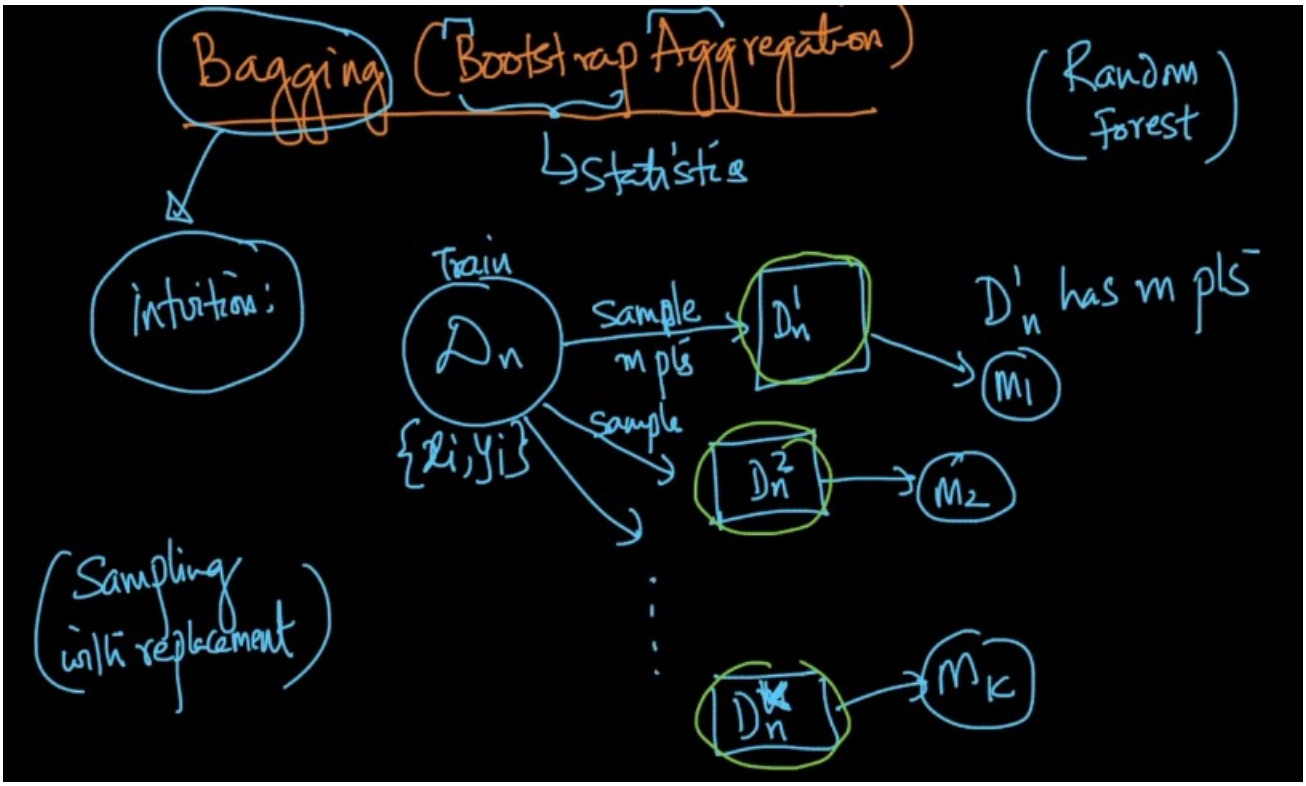
There are four types of ensembles:
1. Bagging. 2. Boosting. 3. stacking. 4. cascading.
The more different the models are the better we can combine them to get more powerful model.



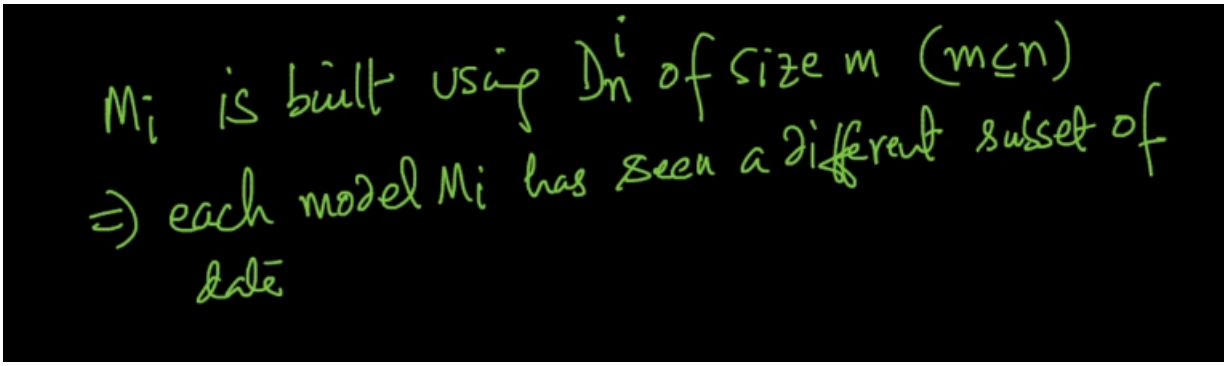


Bootstrapped Aggregation(Bagging) intuition:

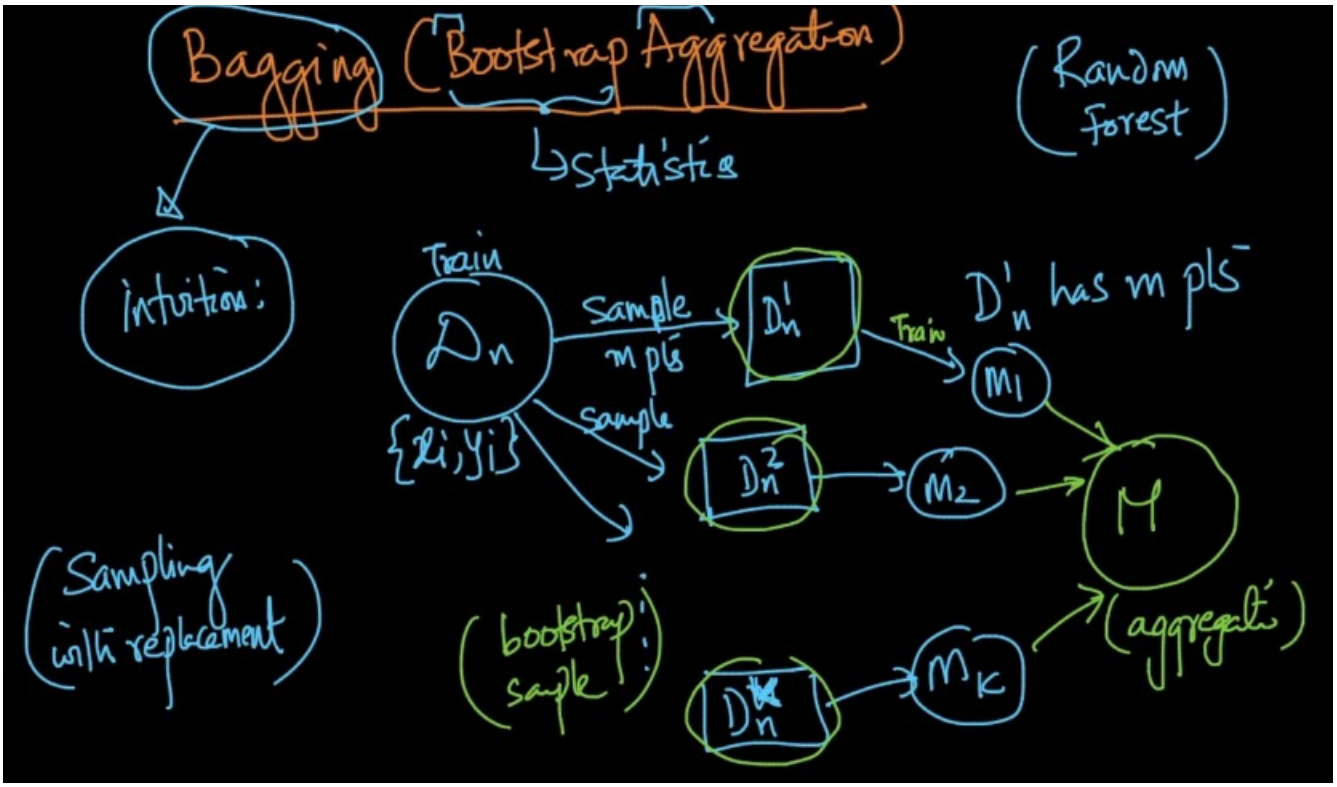
All the samples that are done sample with replacement.



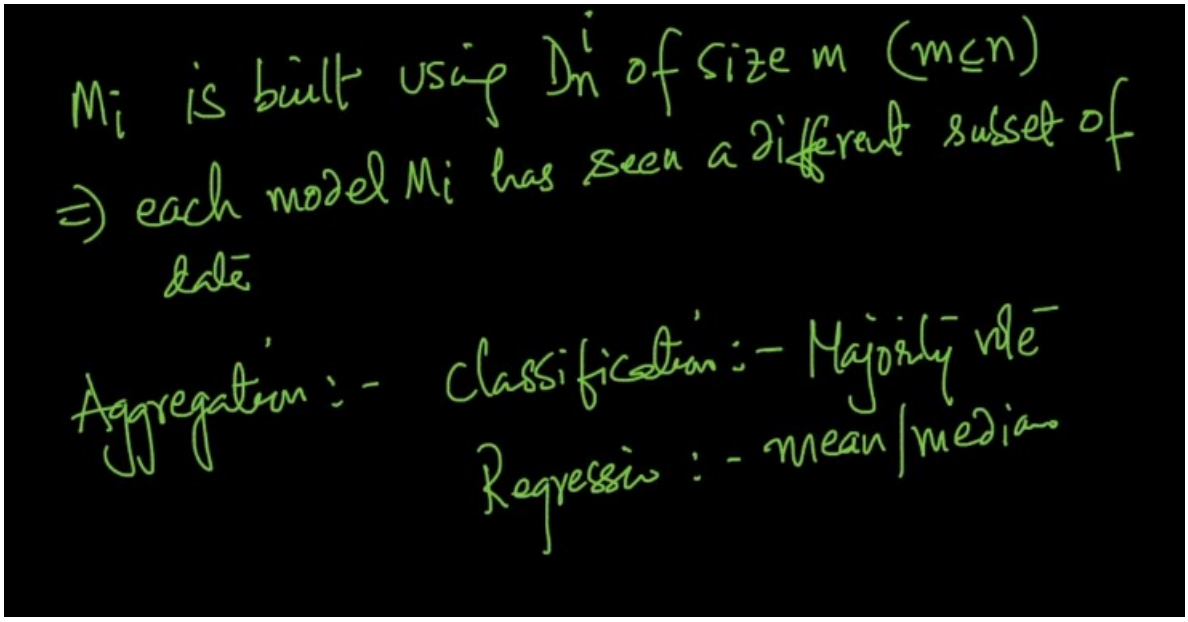
Each model is built on the sample data from the complete data.



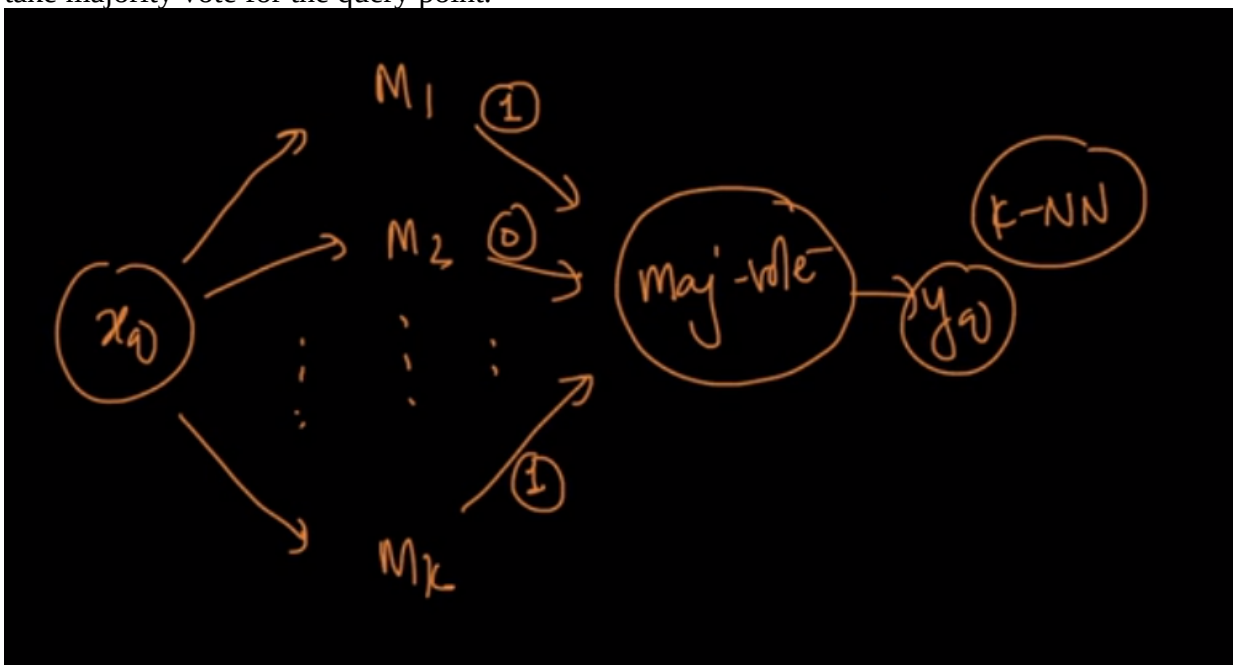
After building all the models we will combine all the models.



The typical way of aggregation is the majority vote in case of classification and mean/median in case of regression.



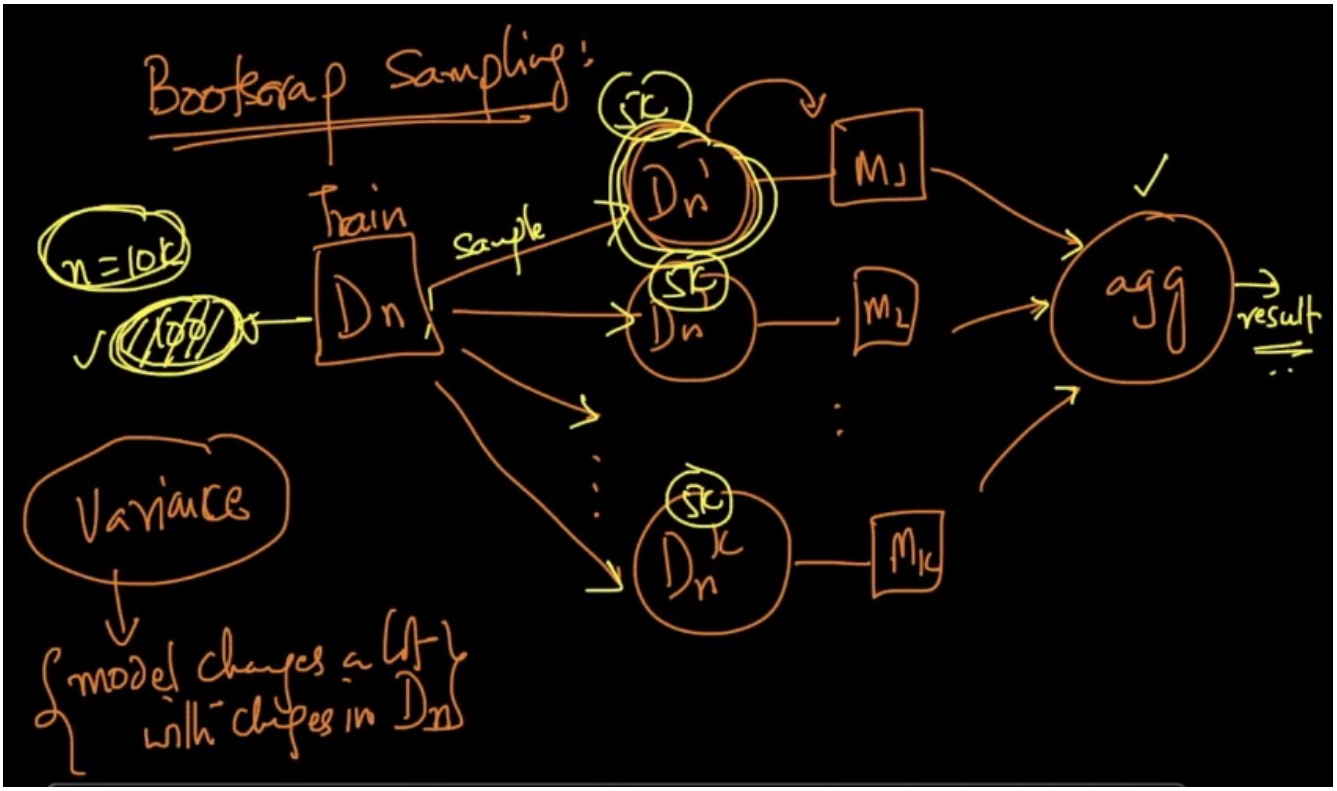
We take majority vote for the query point.



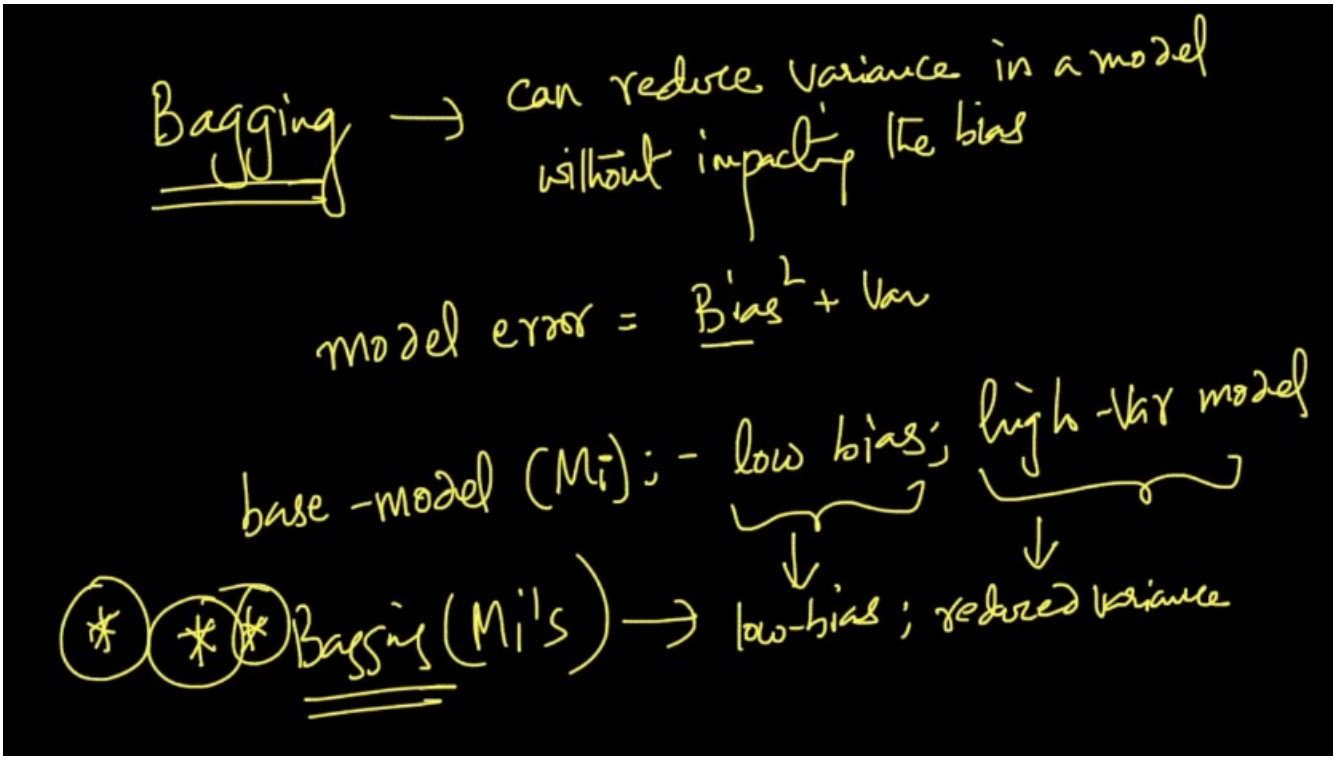
In case of regression we use the mean or median as the prediction.

By changing the original data some of the models will be changed, because every model is developed on subset of the data.

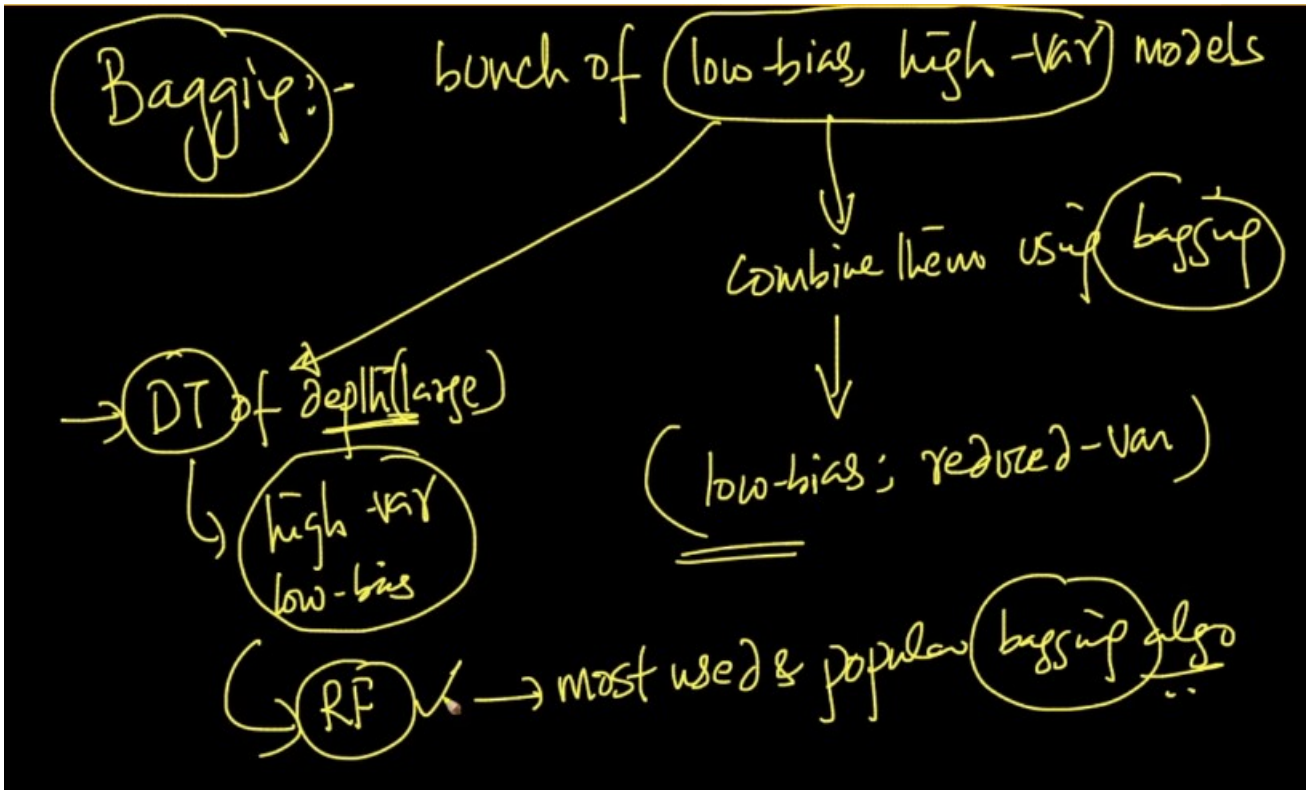
Since, we are choosing aggregation. The over all results does not change.



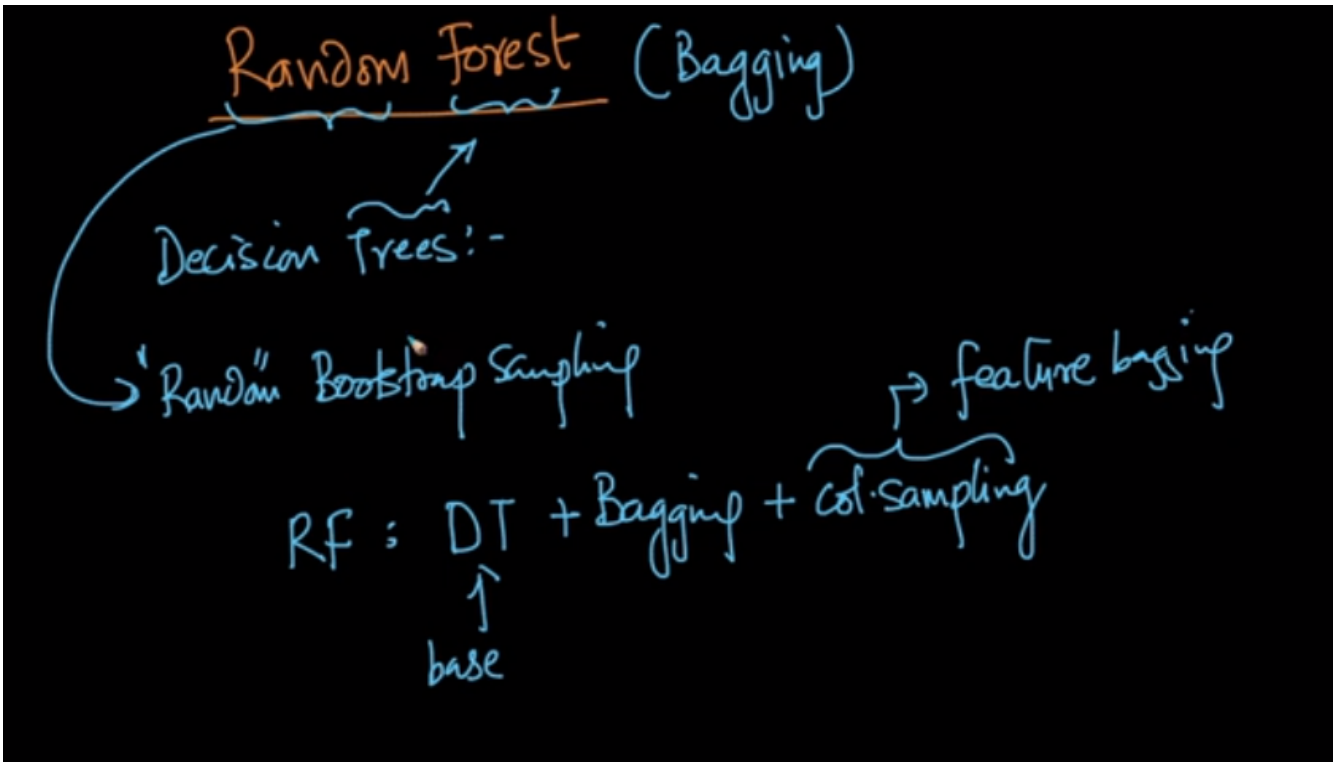
This implies bagging as the model can reduce the variance, without changing the bias.



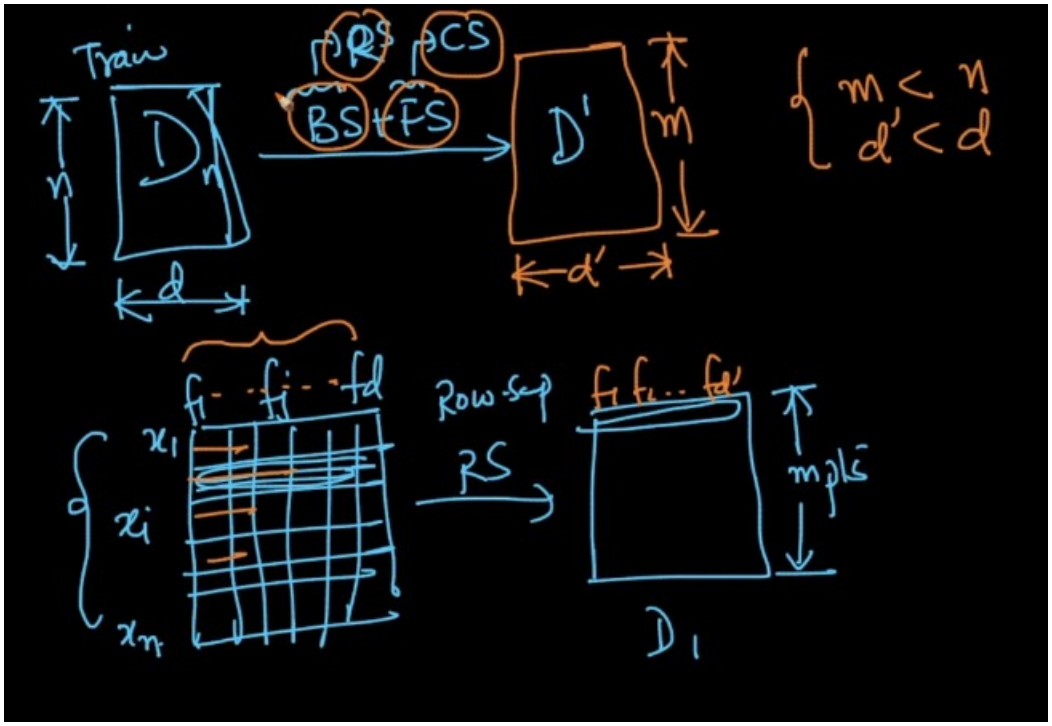
Characteristics of bagging:



Random Forest and their construction:
It takes Decision trees as the base model for bagging and col sampling(Feature bagging).



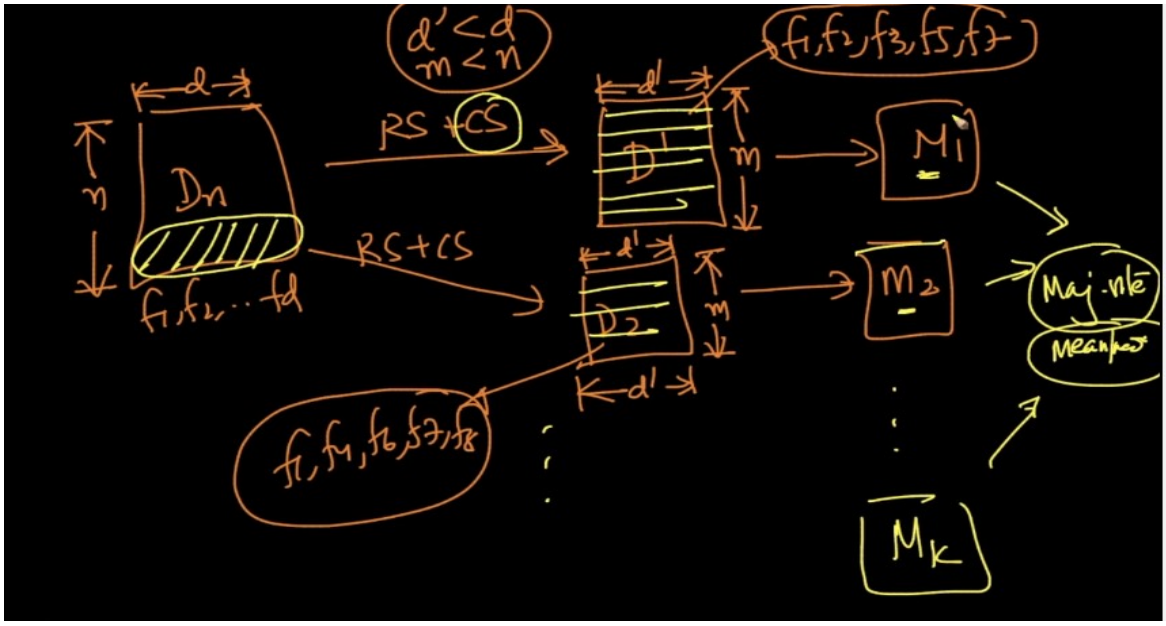
Implementation:



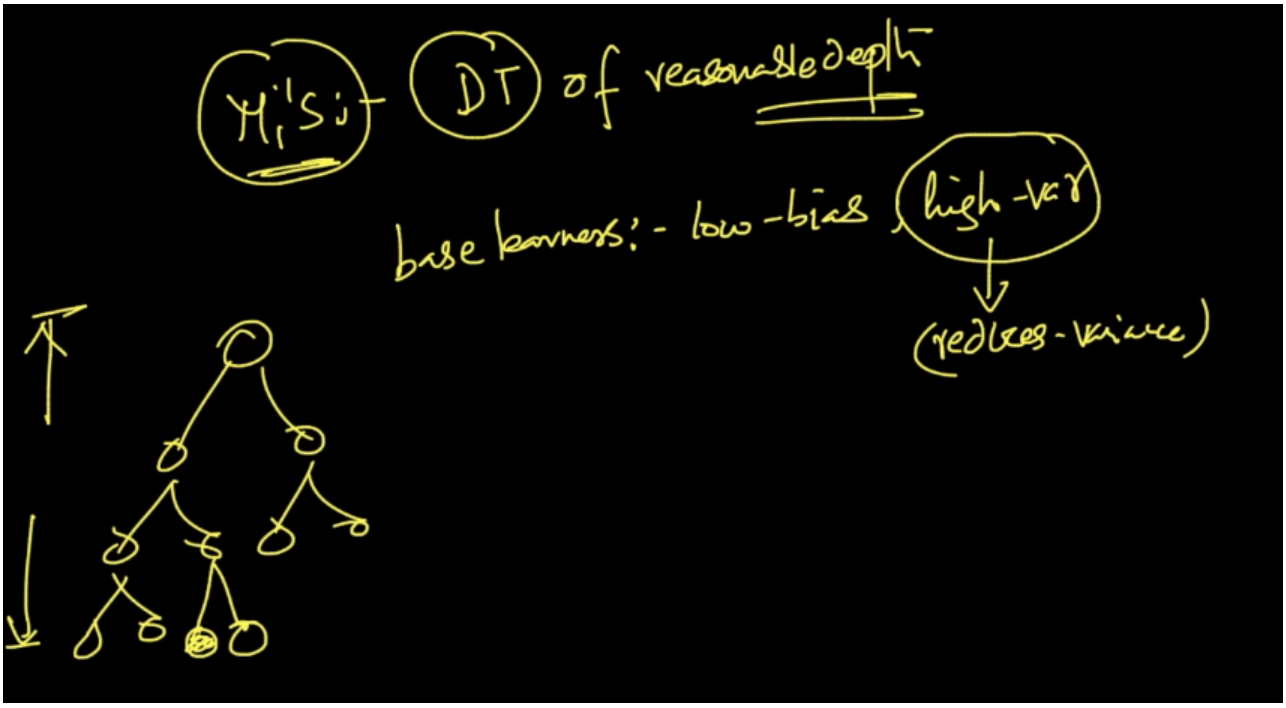
The sampling of rows is done randomly with replacement.
The selected sampled data set dimensions(or features) are also selected randomly.

The model M1 is made on this data.
The rows and columns are also sampled randomly in Random Forest.
The above process is done for making several other models. The data sets will be more different than the bootstrap sample as the features selected are different.

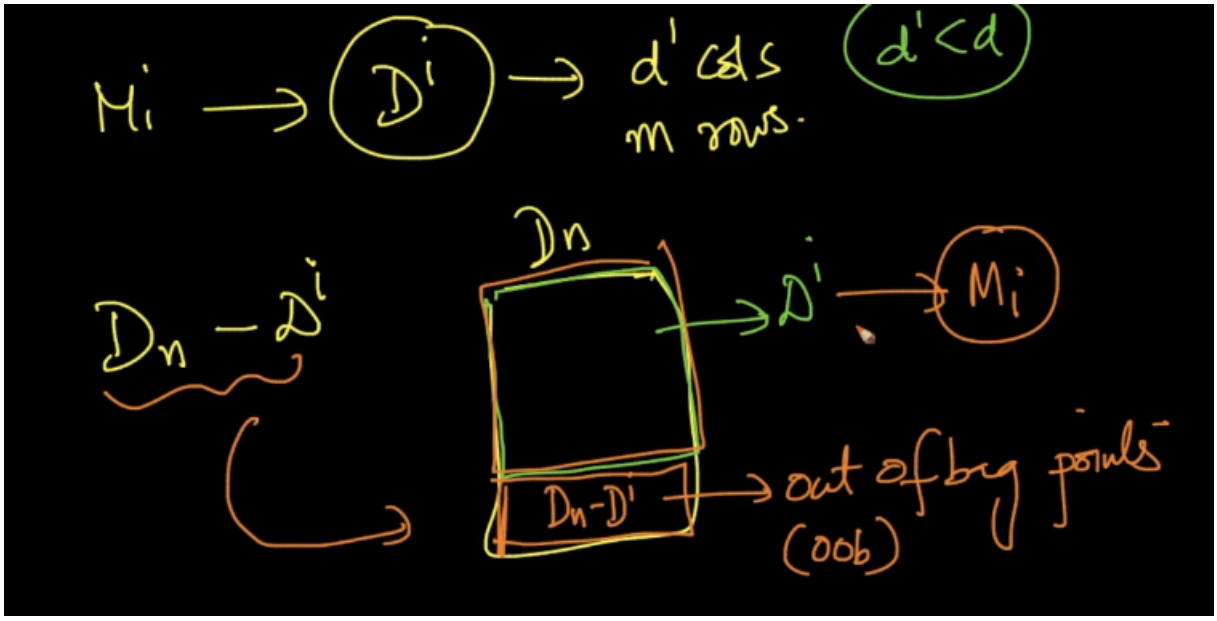
At the end we do majority vote.



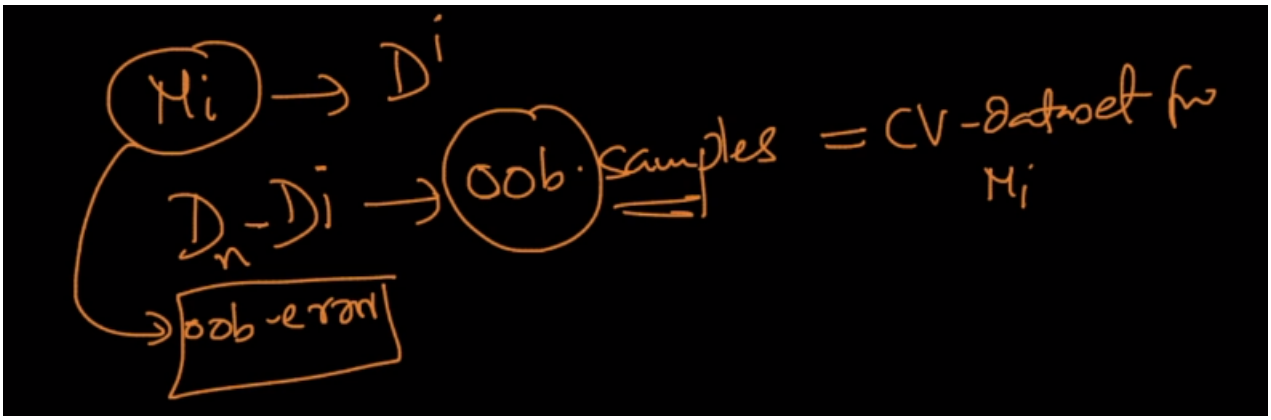
The Mi's have no limit on the depth, making each models to over fit(high variance) does not impact the over all performance as the majority vote is done to reduce the high variance(over - fitting) of each models.



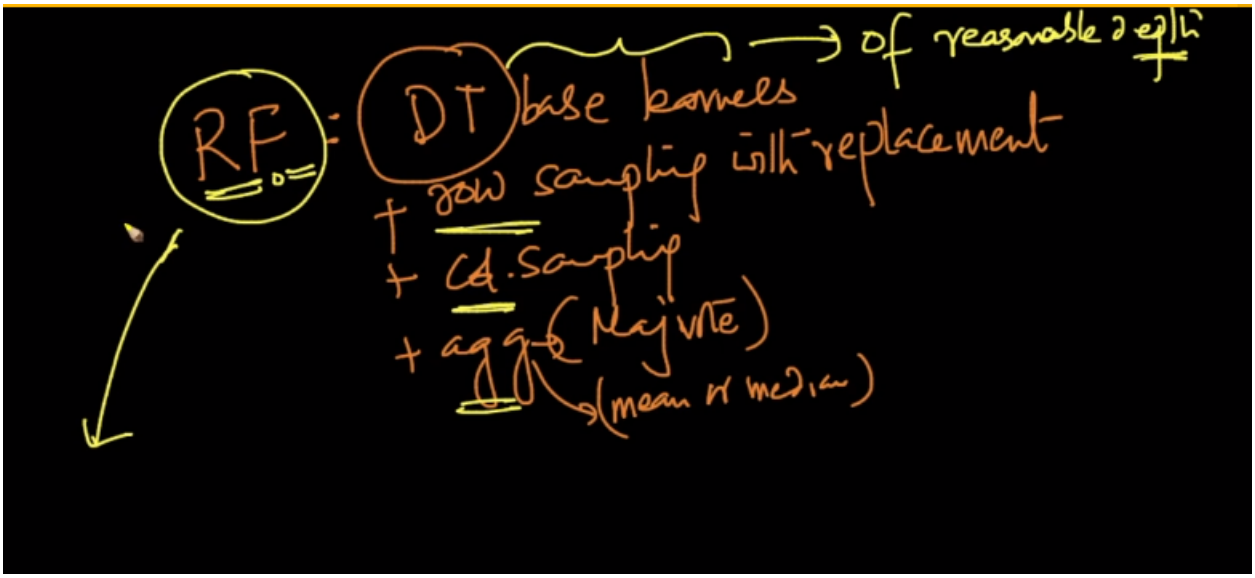
The out of bag points are nothing but the points that are not included in the sample data for training the model. These **OOB** can be used for cross – validation.



This error is called **OOB** error.

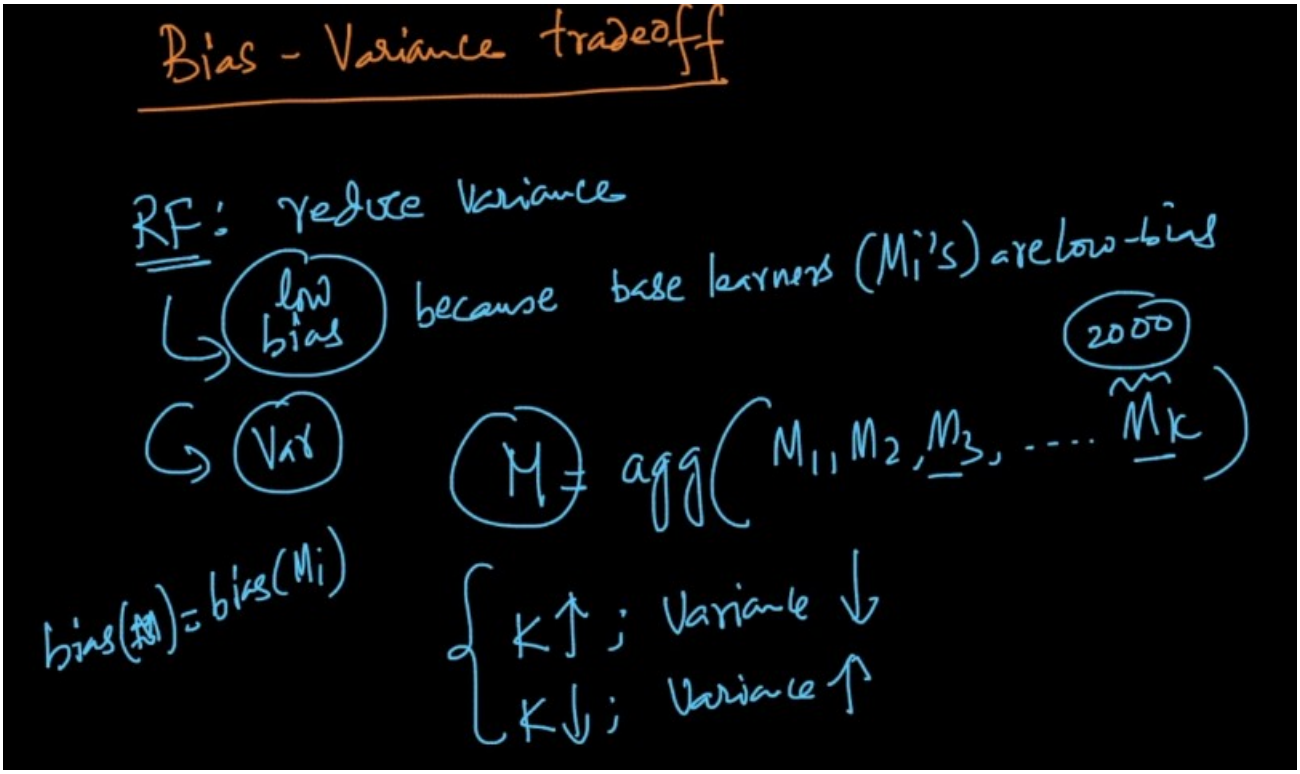


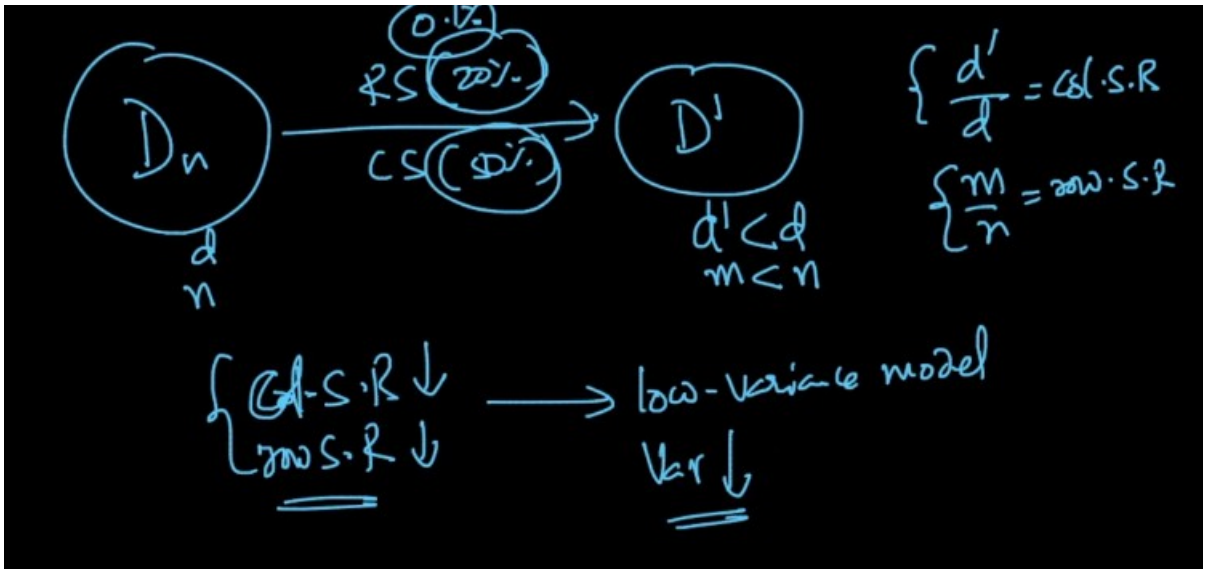
Summary:



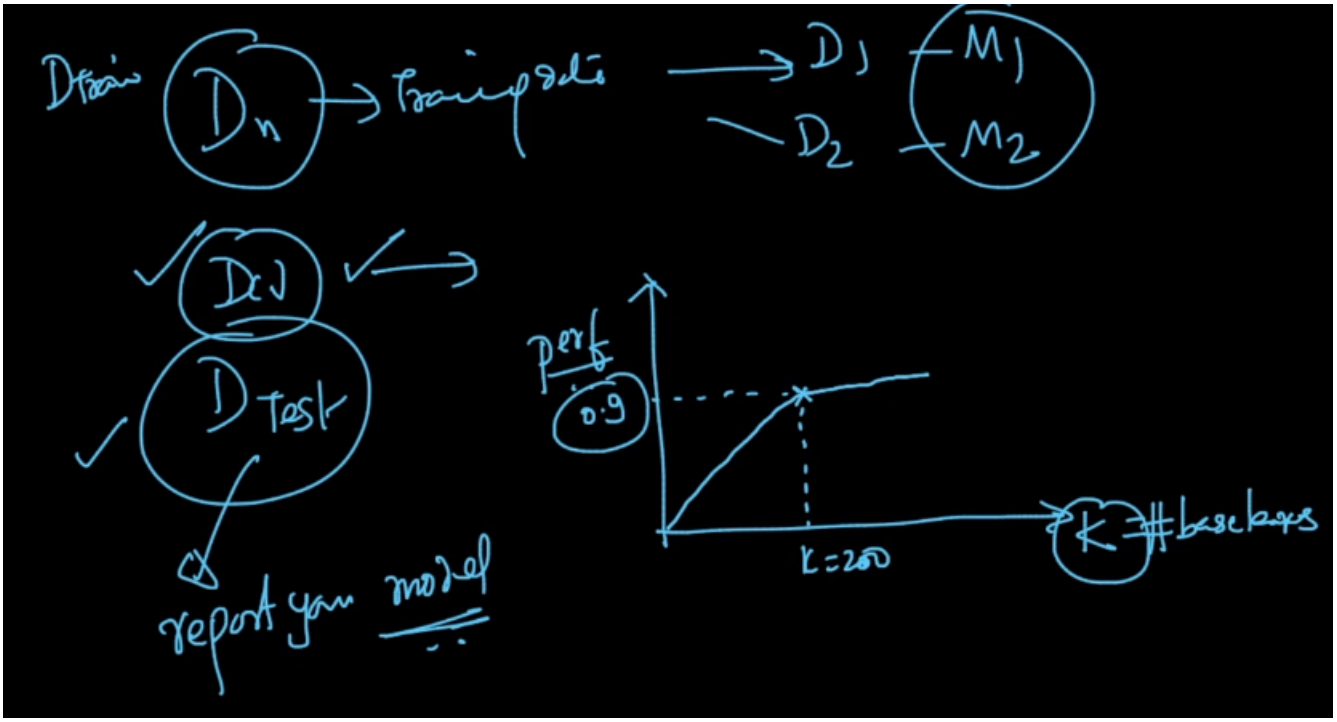
Bias – Variance trade off:

Random forest are said to have less bias because the base models are low – bias.



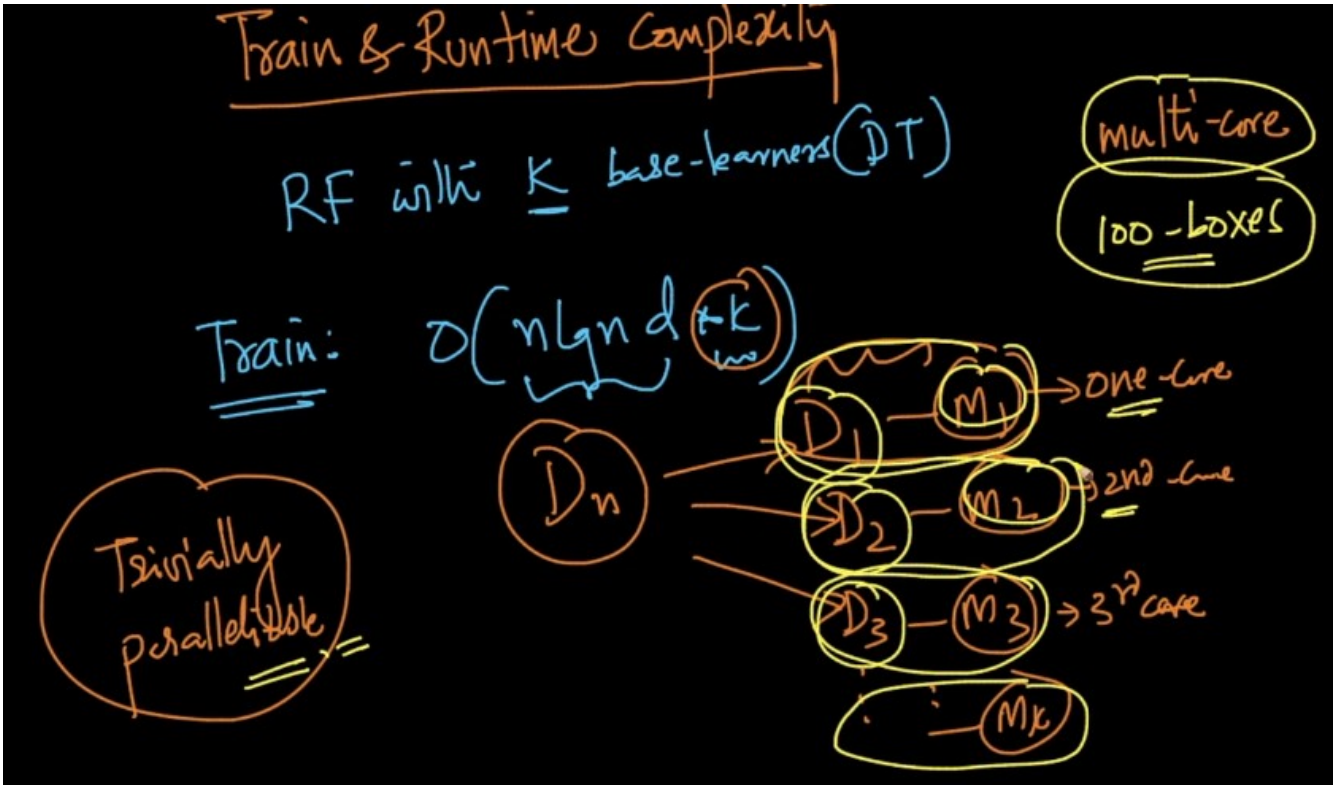


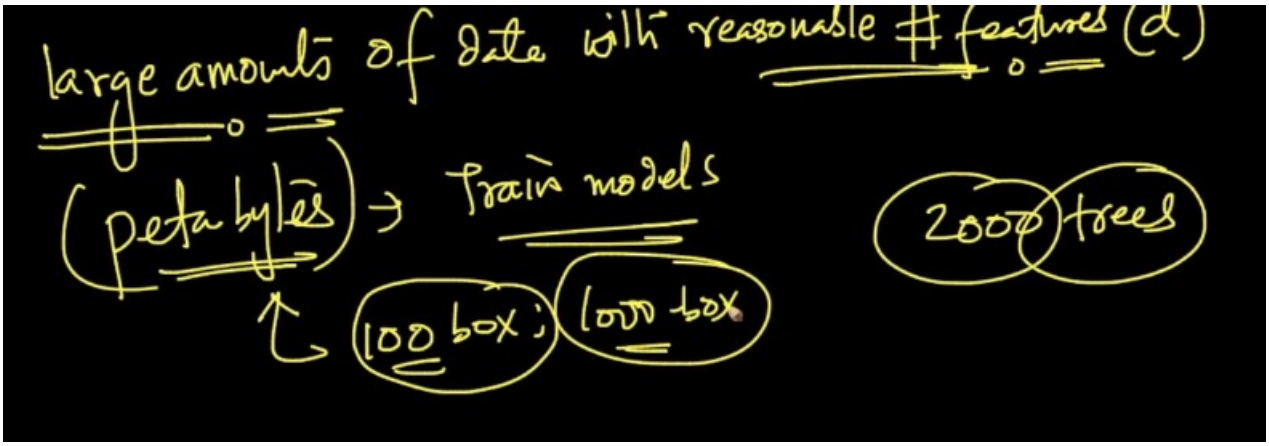
Often times people tend to keep the column and row sampling rate constant and keep increasing the models.



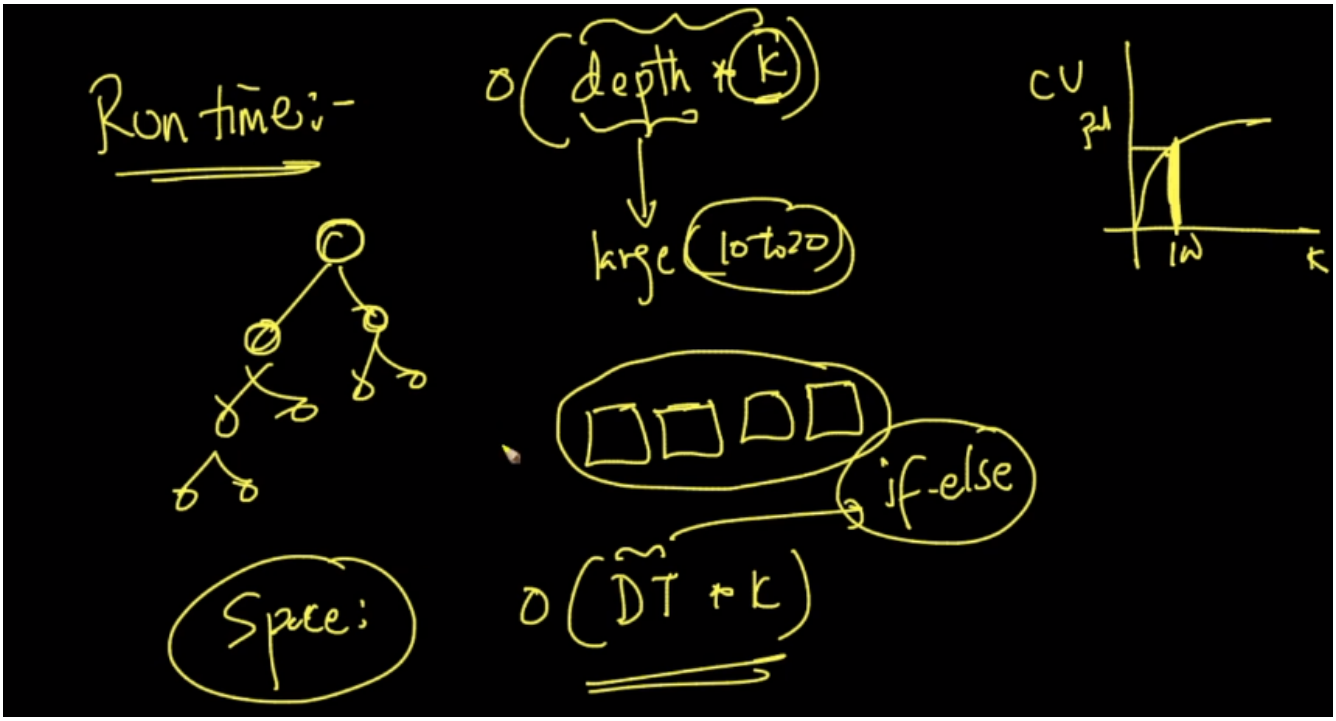
Cross – validation is used to choose the hyper parameter K(number of base learners).
The number of base learners is the important hyper parameter than the row and column rate.

Bagging: Train and Run – time Complexity:
The complexity of a decision tree is: n*log(d)
Each of the models can be trained independently. Random forest is trivially parallizable.





Run time complexity:
The model it self can be made into if – else condition script.

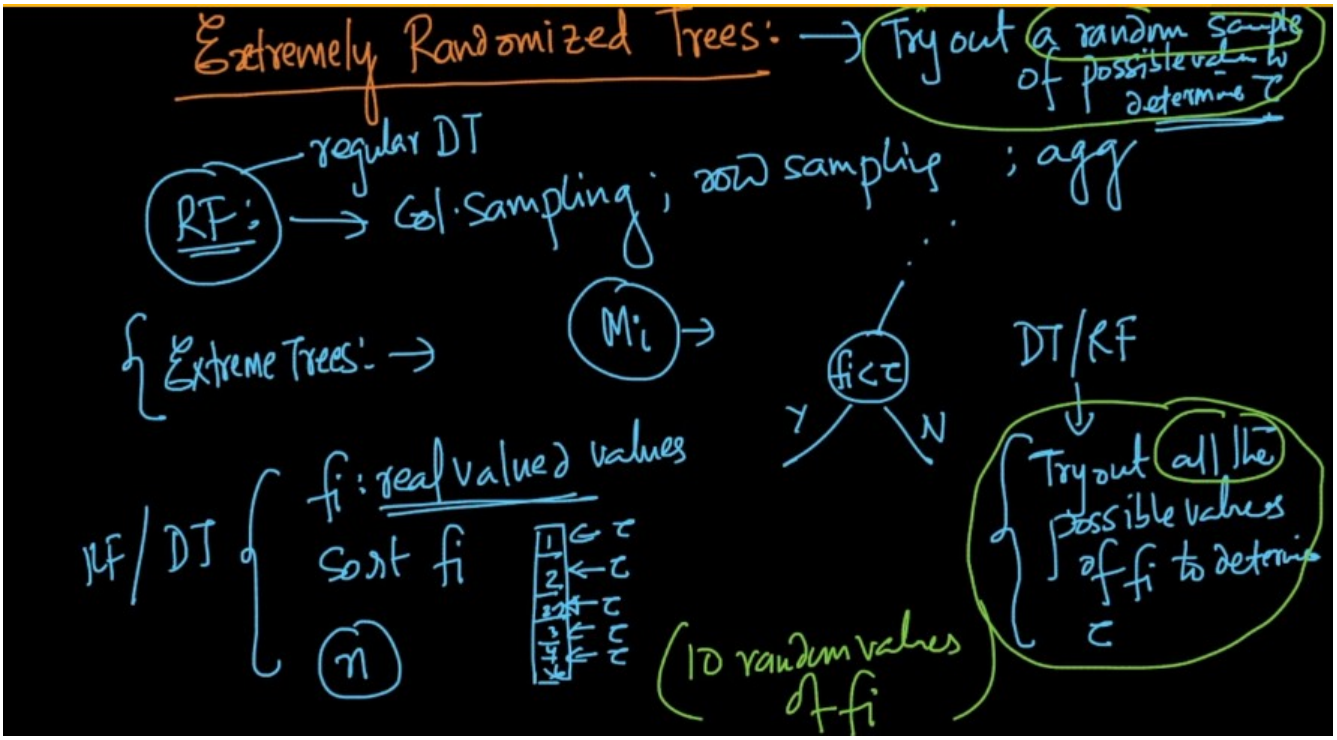


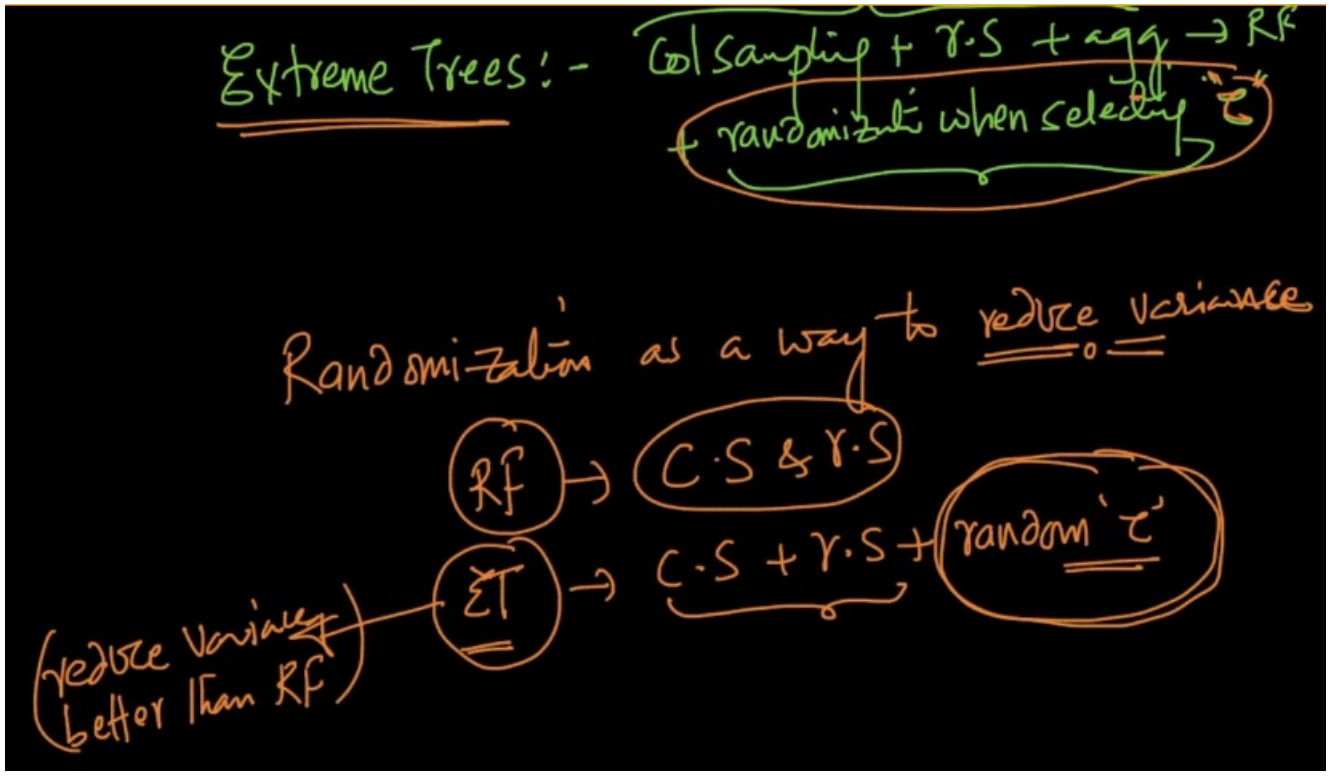
The run time and space complexity of Random forest is very low.

Extremely randomized trees:

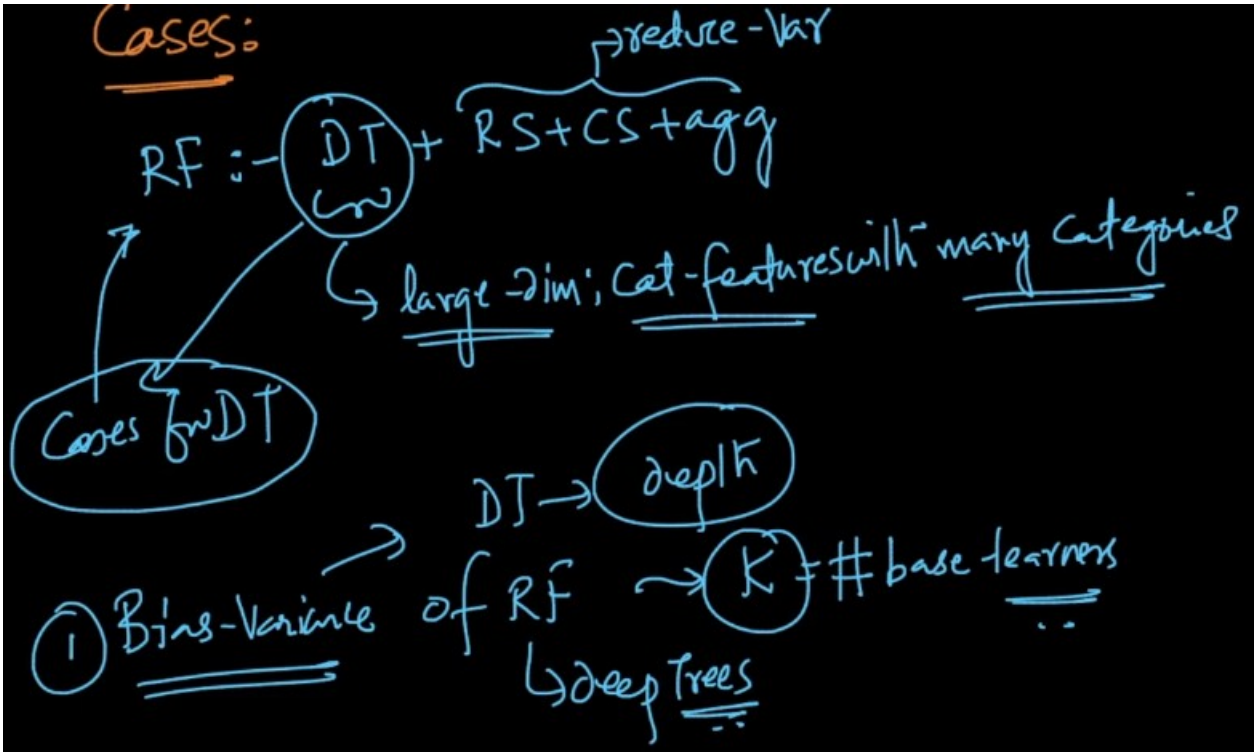
In regular decision trees, we will try out all the values of the threshold and get the best possible value of the threshold for real – valued features, that increase the information gain.

In randomized decision trees the some values of the bag are selected randomly and then we come up with a threshold. This is our final threshold of the base model.


Extremely Randomized Trees: → Try out a random sample of possible values to determine 'c'
RF: → regular DT
Col. sampling; row sampling; agg
Extreme Trees: →
Mi →
fi < c
Y
N
DT/RF
Try out all the possible values of fi to determine c
RF/DT
fi: real valued values
Sort fi
n
(10 random values of fi)

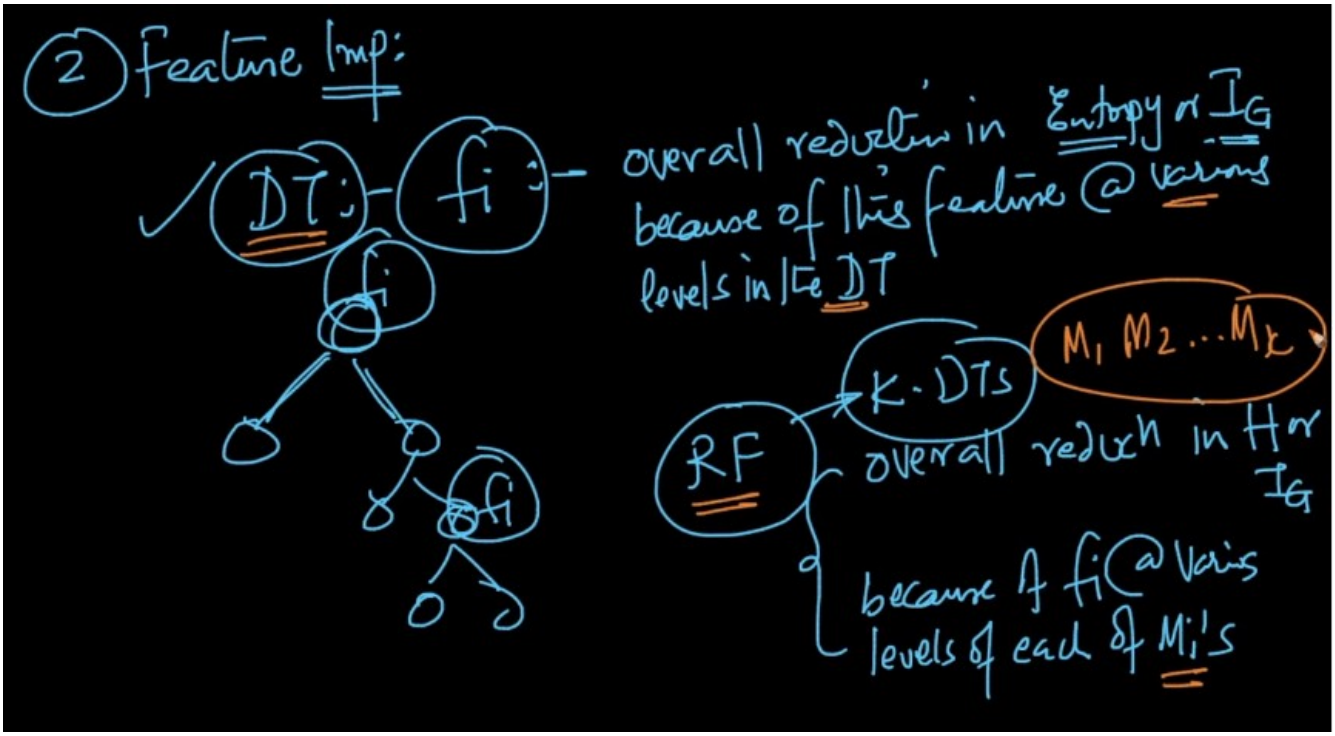

Extreme Trees: - Col sampling + r.s + agg → RF
+ randomization when selecting 'c'
Randomization as a way to reduce variance
RF → C.S & r.S
ET → C.S + r.S + random 'c'
(reduce variance better than RF)

Cases:



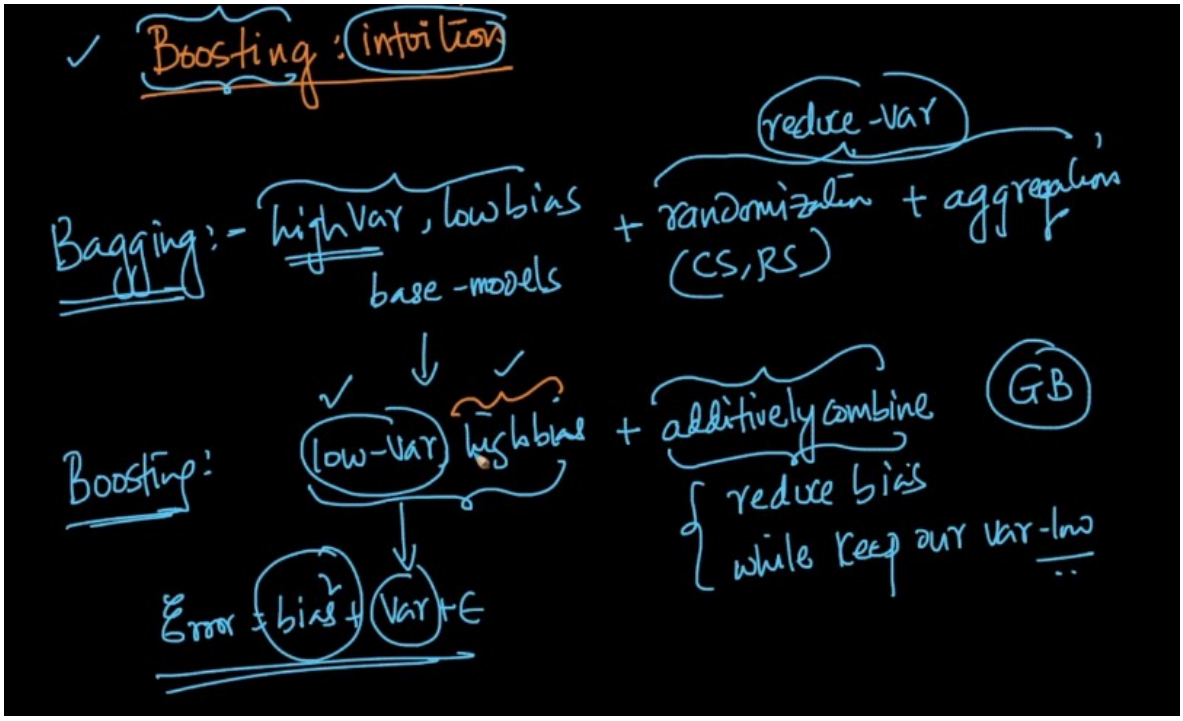
Feature importance: The overall reduction in entropy because of this feature at various levels in DT of all the base models.

Then the feature is said to be more important.

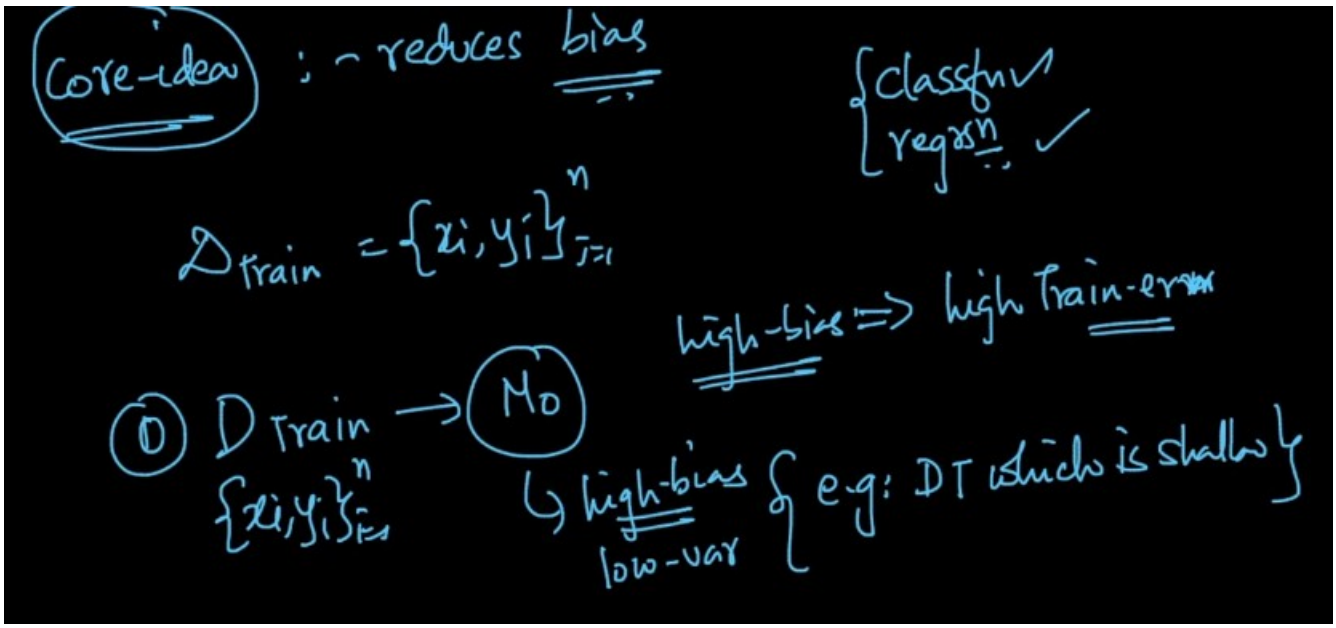


Boosting Intuition:

Instead of making high variance and low bias models in bagging, we use low variance and high bias models in Boosting with additive combine, that will reduce bias while keep out variance low.



High bias can be viewed as high training error.



Step – 1:

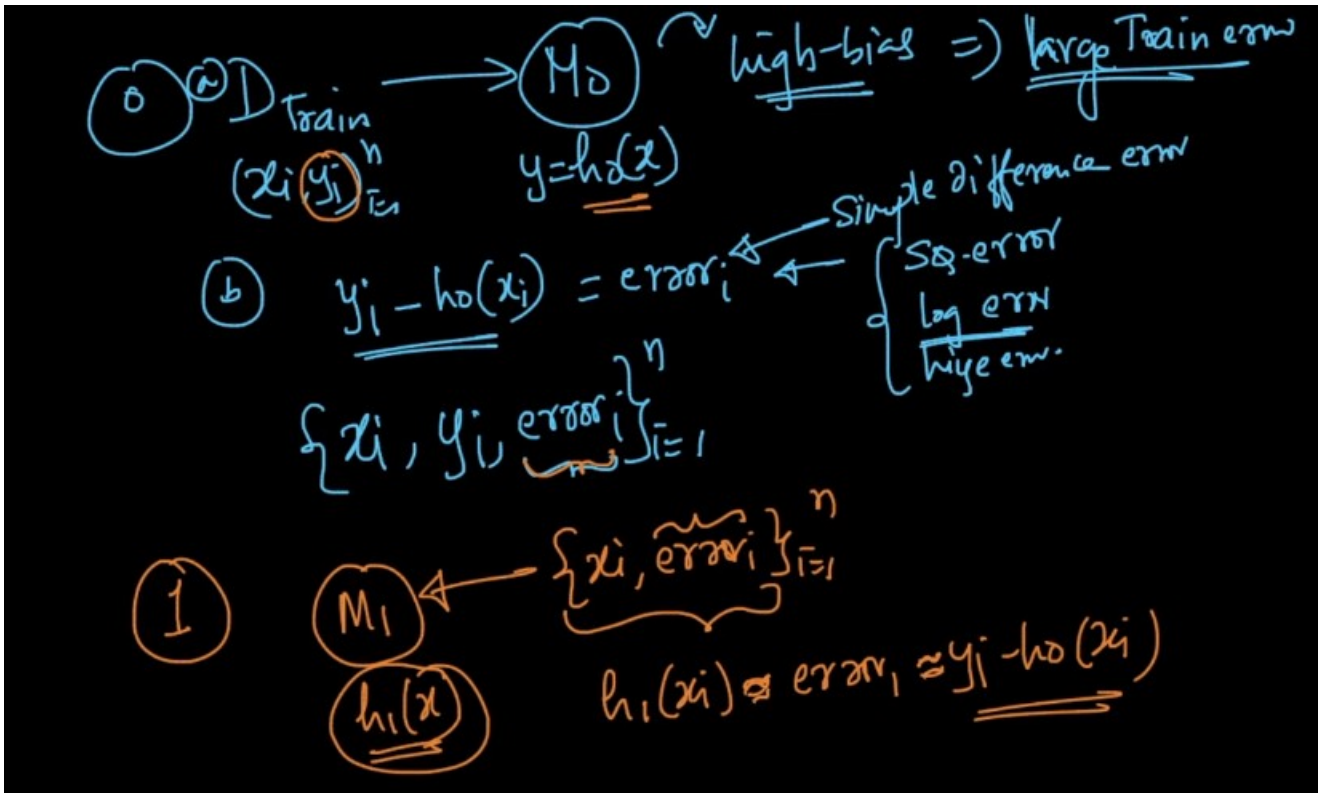
Train the model on all the data points. Make the model h(x).

Step – 2:

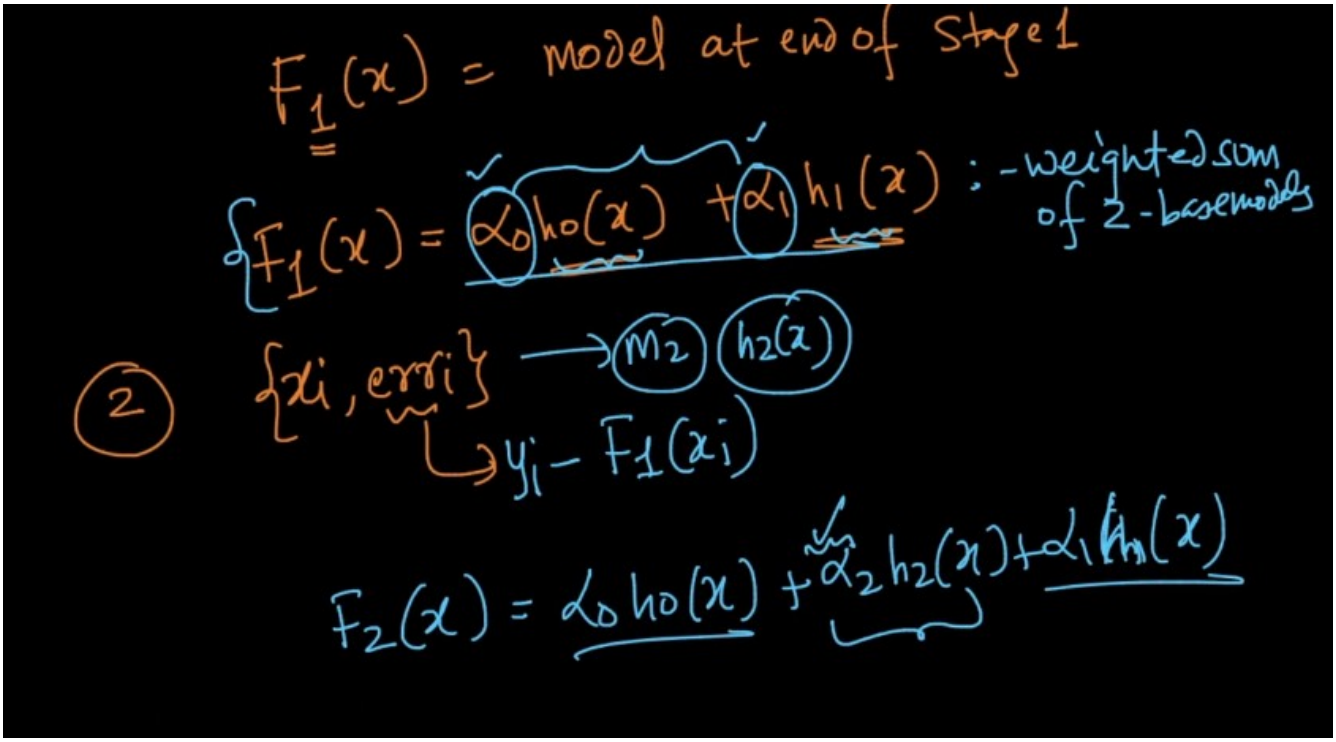
Get the error on the points. Yi – h(xi). This is a simple error.

Step – 3:

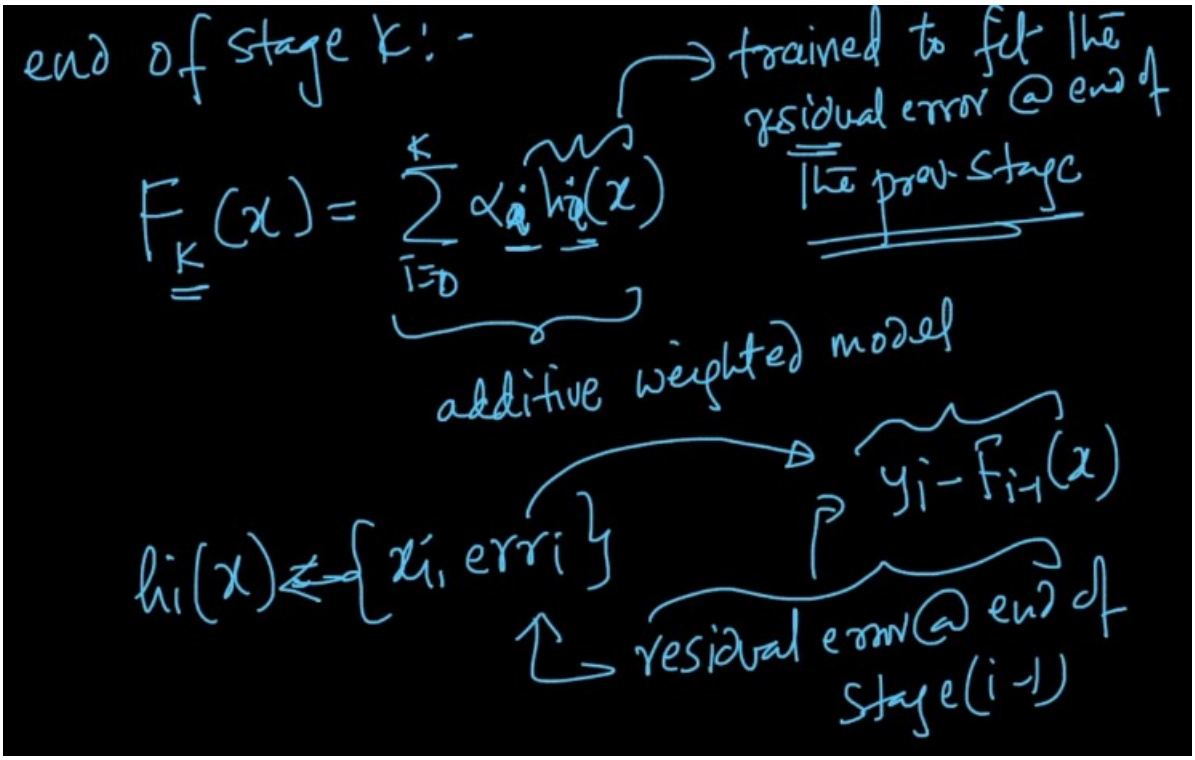
Train the model on the error that we got from the previous model.



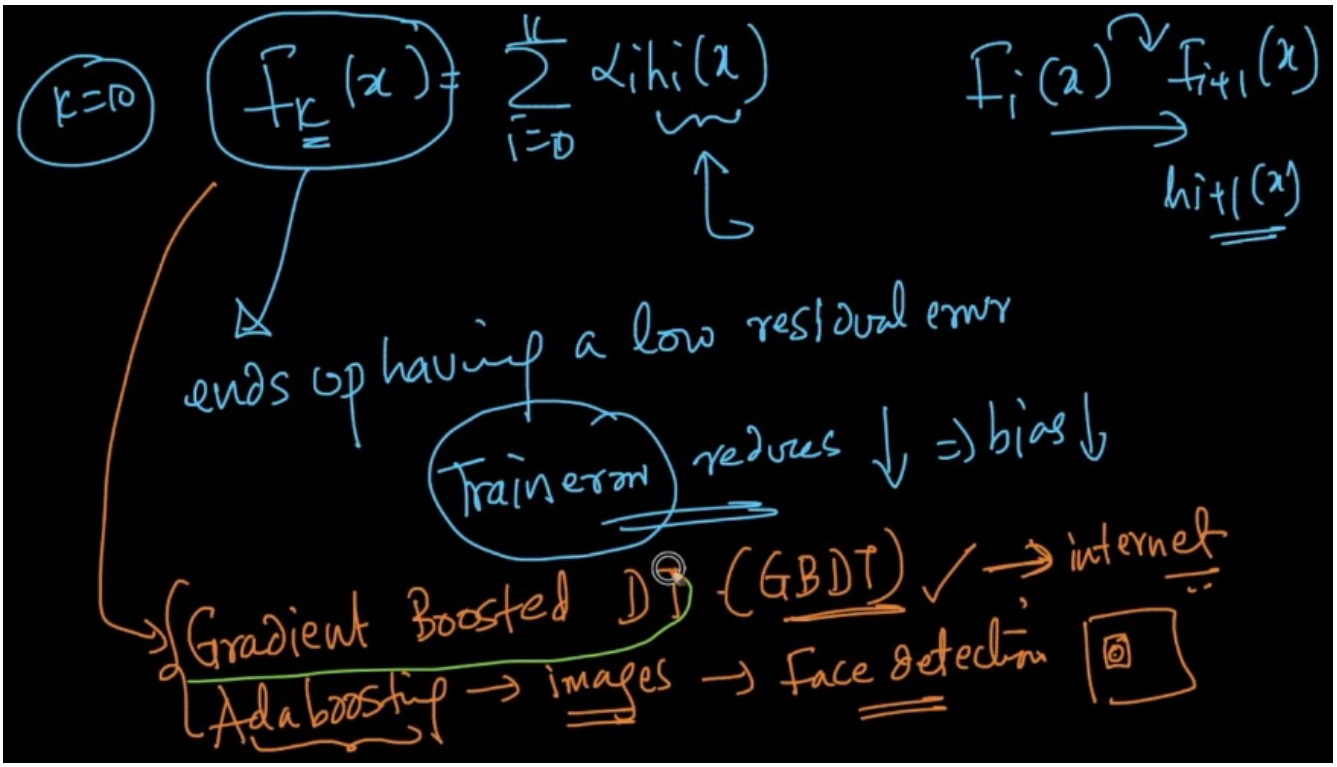
The models are made by additively combining the models.



Generalized notation:

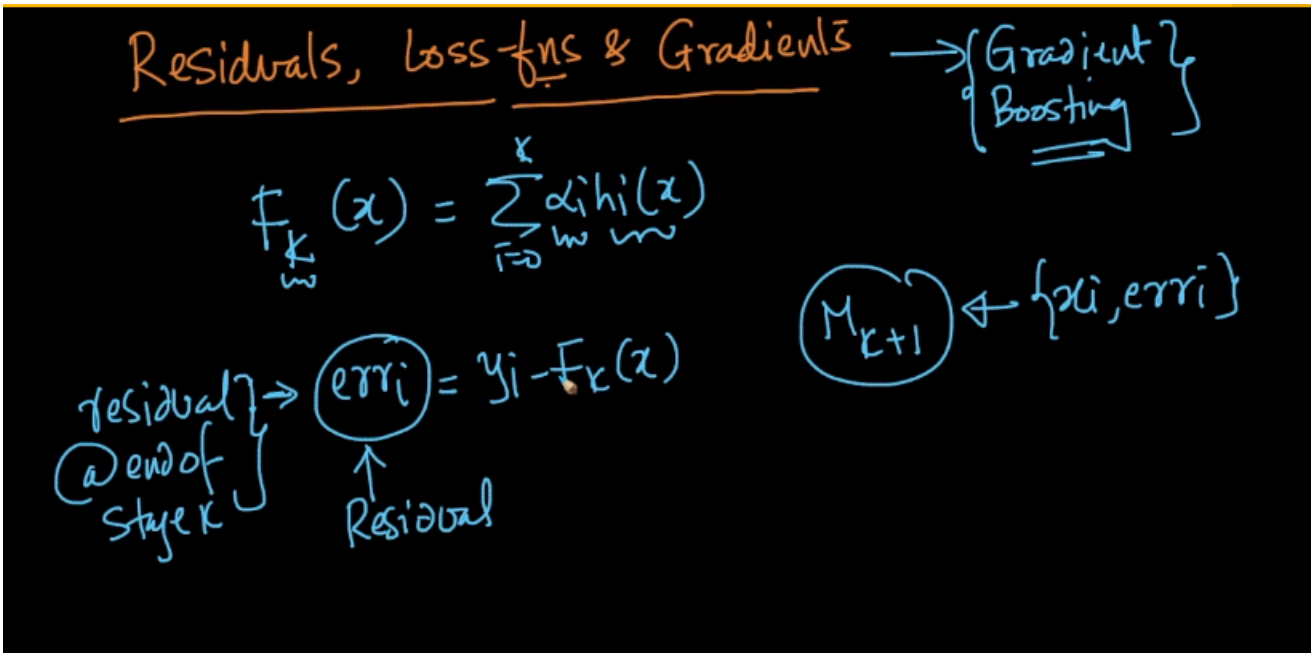


The every model that is being constructed is to reduce the high error/bias. At every level the bias/error is reduced.



Residuals, loss functions and gradients:

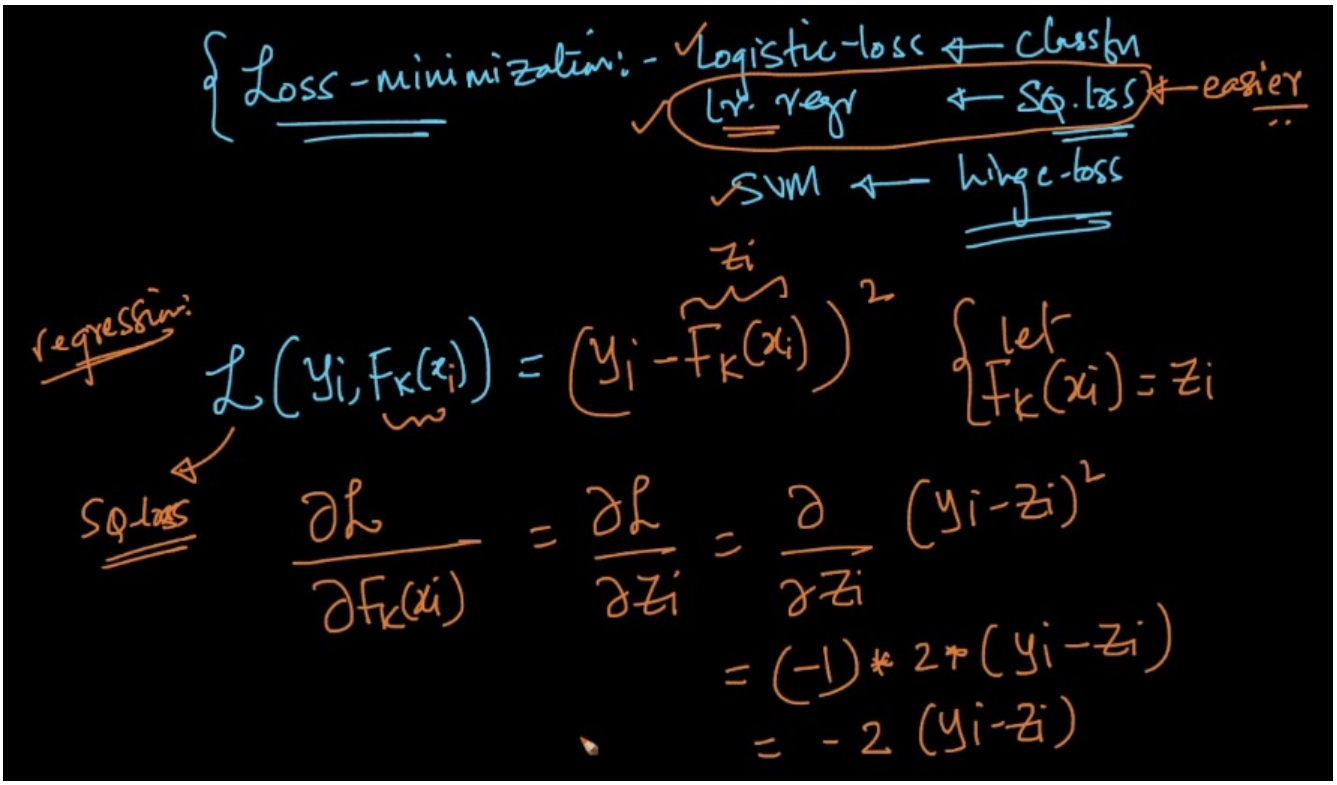
Residuals:



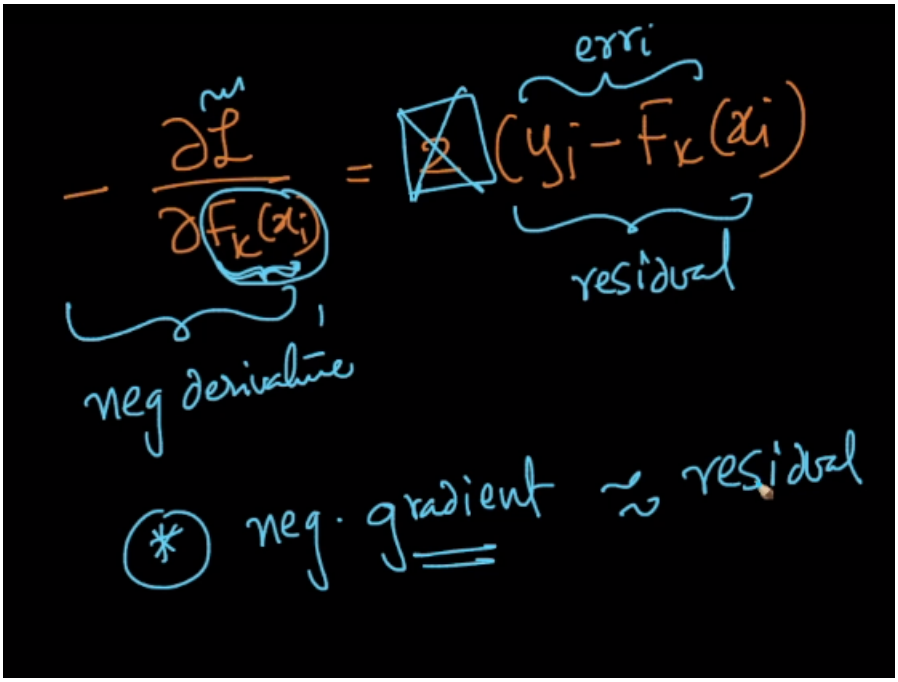
Here the goal is to find the <alpha> values and number of base learners. How to model the base learners.

Loss minimization:

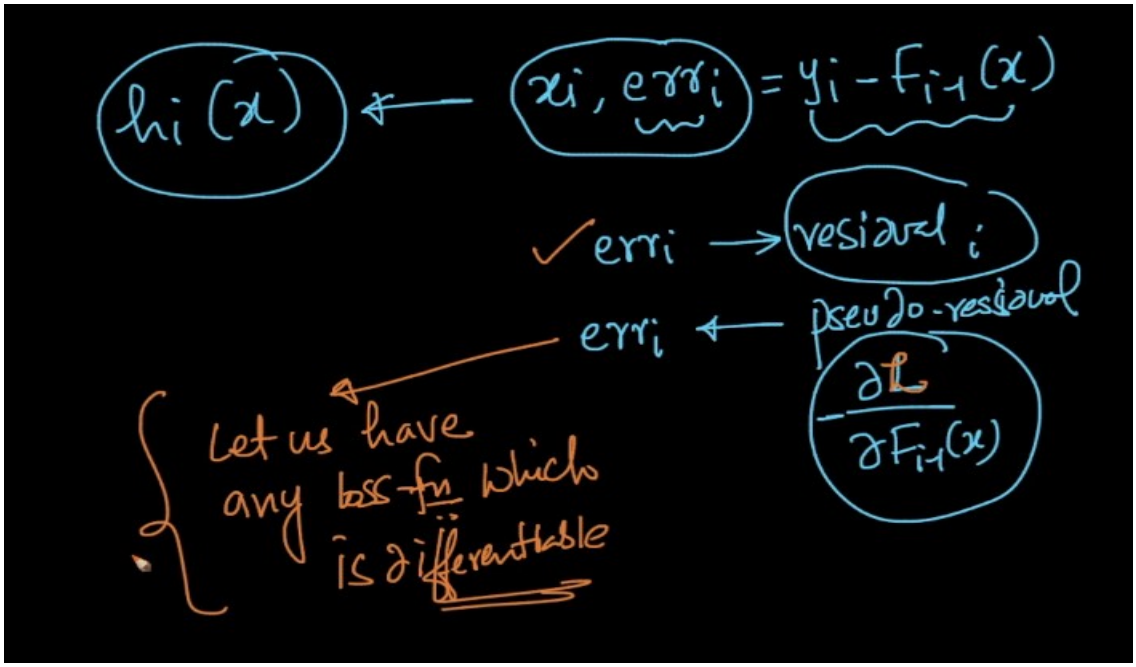
For one base learner.



At the end of the ‘K’ th base model.
The error is called the residual error. The negative gradient can be taught like it is proportional to the residual error.

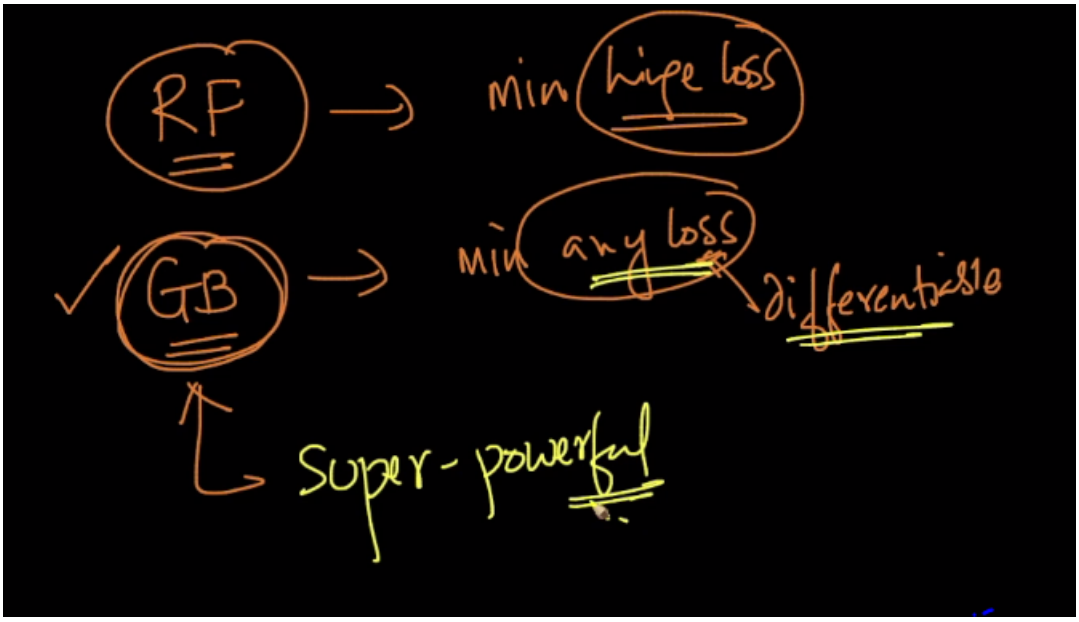


This is often called as pseudo residuals.

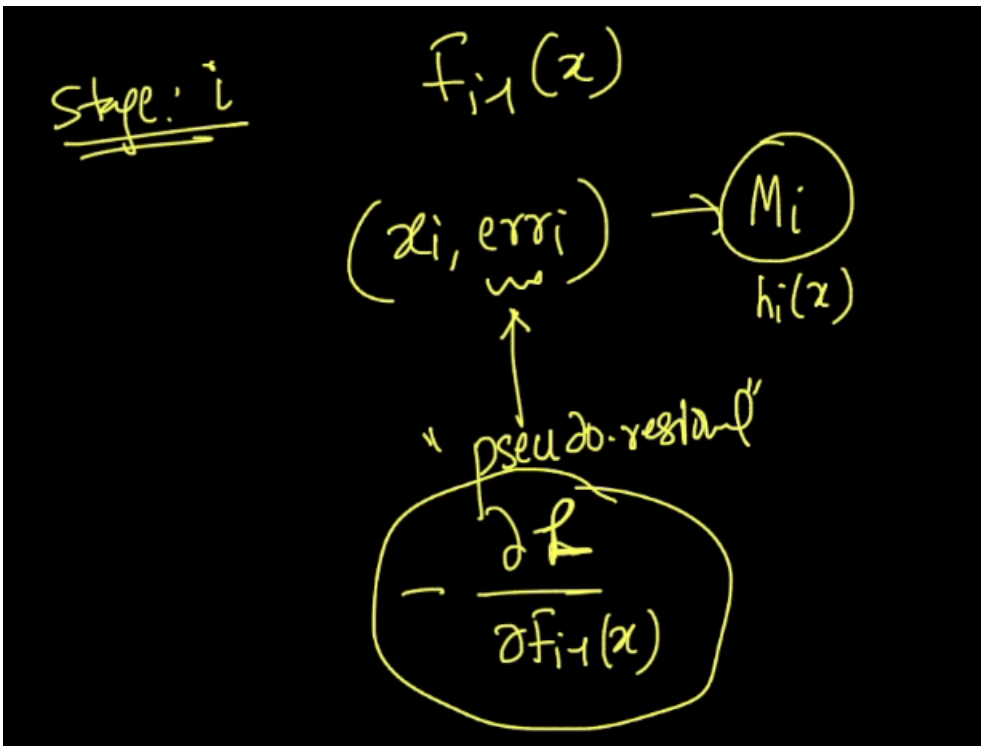


Replacing the residuals with the pseudo residuals, will help in replacing with the any loss function which is differential.

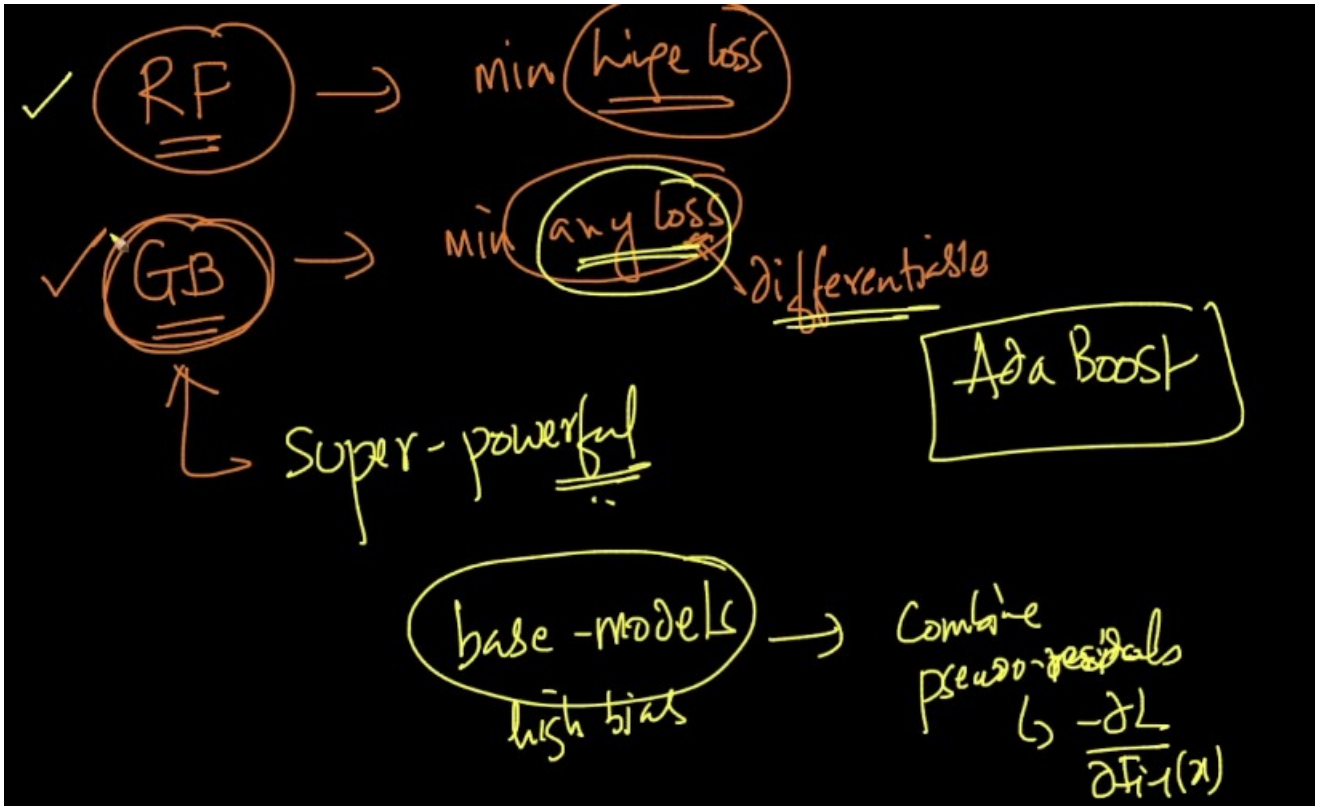
As long as the loss function is differential we can replace with any other loss function.
This is the reason gradient boosting is so popular.



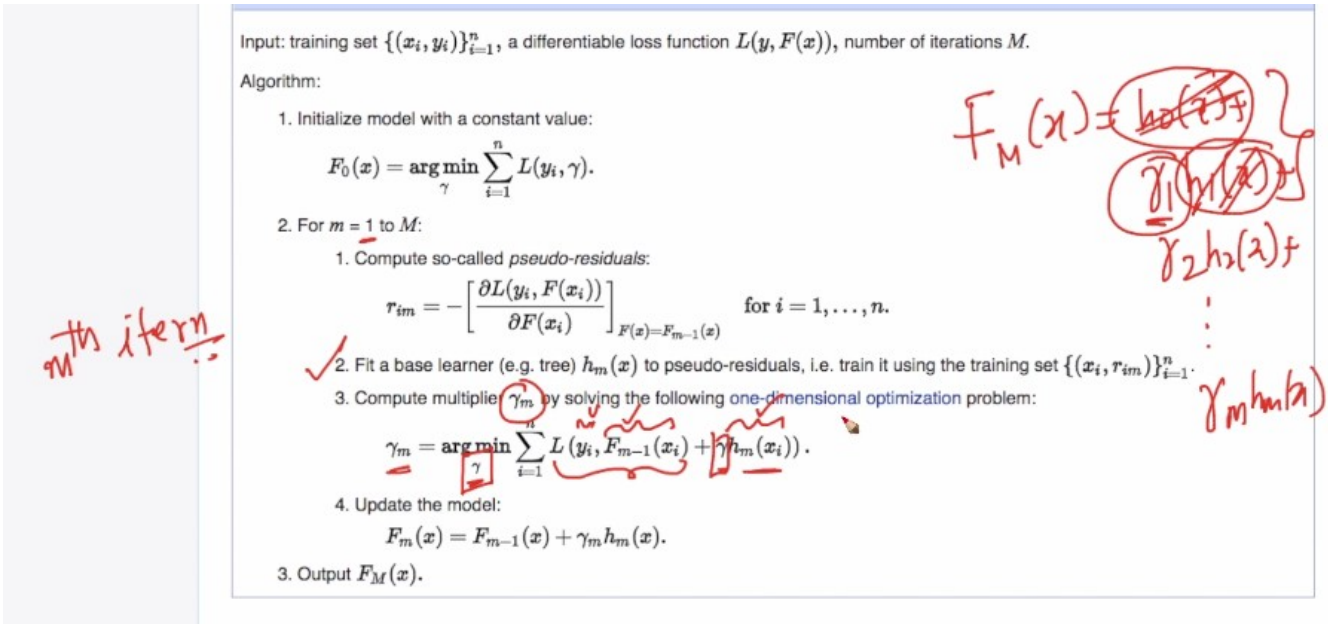
At any stage ‘i’ we can take the pseudo residual of the actual error got from the previous level.



We have the other algorithm called Ada-boost.



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:

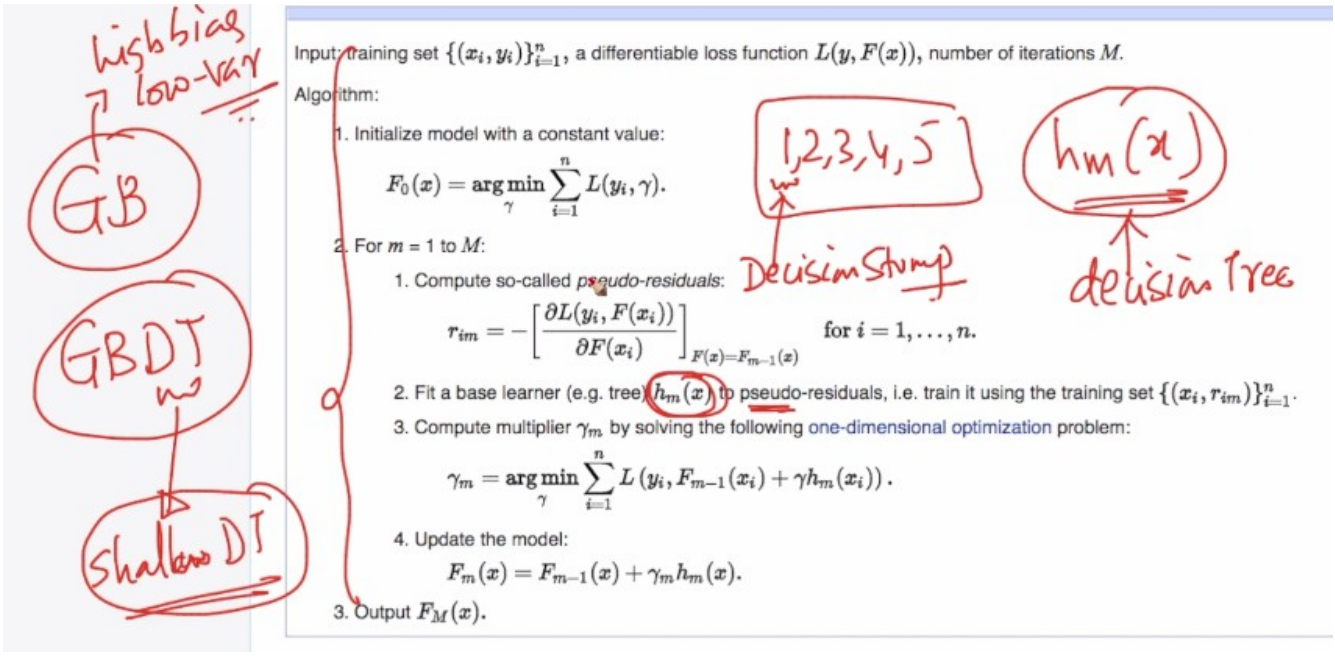
$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

   4. Update the model:

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

3. Output $F_M(x)$.

Gradient boosting is the general idea. The each of the model must be high bias and low variance model. This can occur in case of decision trees with shallow depths.



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*:

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

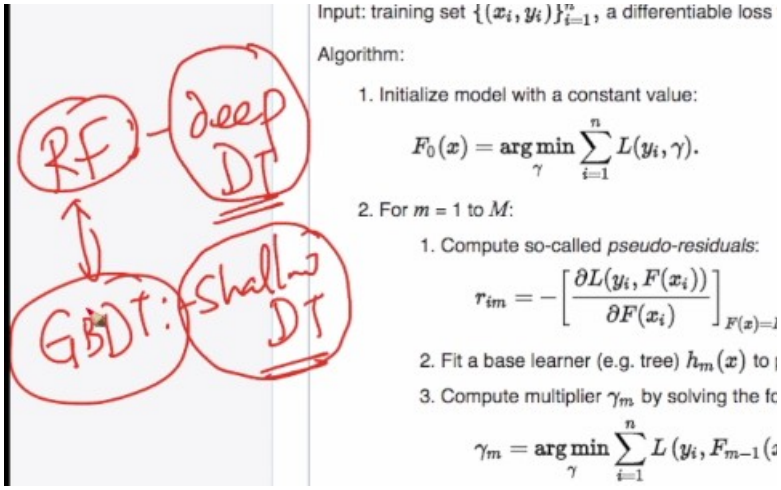
   2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
   3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

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

   4. Update the model:

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

3. Output $F_M(x)$.



Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss

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*:

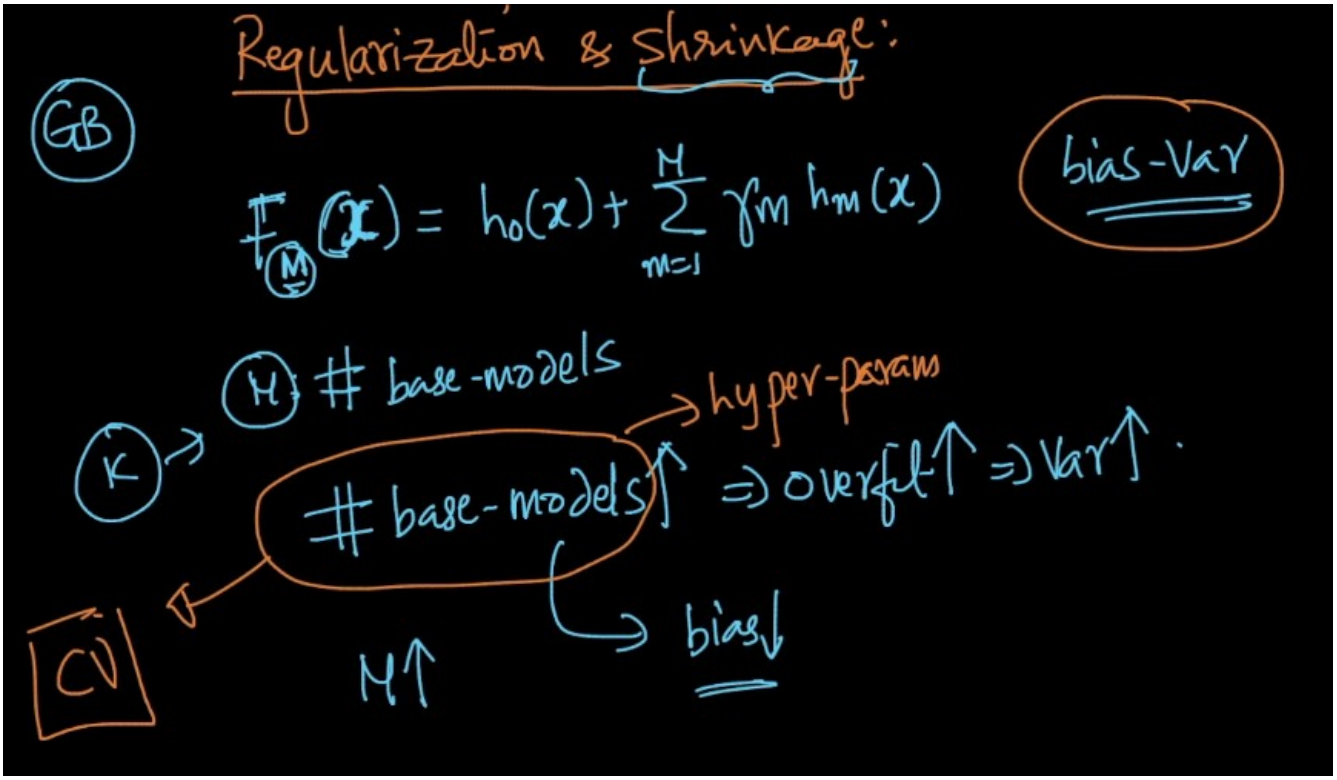
$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=}$$

   2. Fit a base learner (e.g. tree) $h_m(x)$ to
   3. Compute multiplier γ_m by solving the fo

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

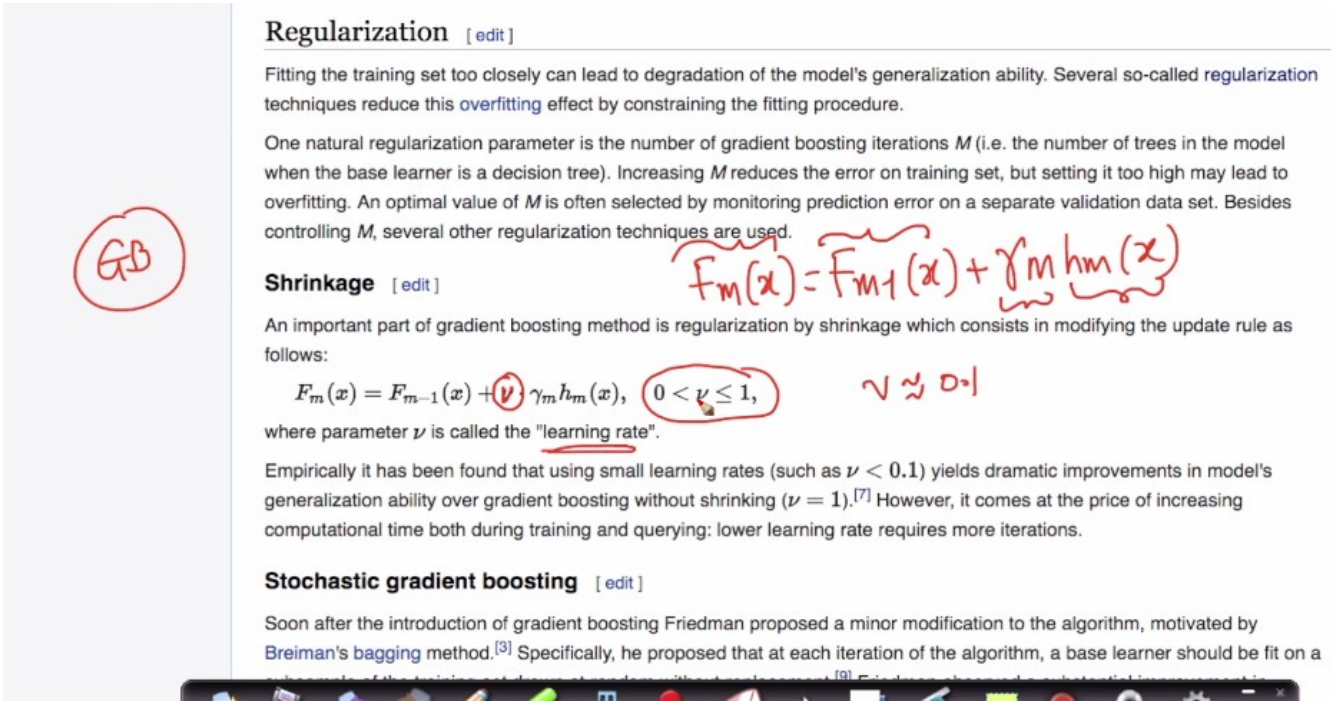
Regularization and shrinkage:

As the number of base models increase then we would over fit and variance can also increase, bias can decrease.



The number of base models are the hyper parameter, that can be chosen by cross – validation.

Here the shrinkage can be tuned by using the variable ‘V’, which is called the learning rate. Lies between 0 and 1.



Here ‘V’ is also a hyper parameter.
If ‘V’ is very small then the chances of over fitting also reduces and variance also decreases.
If ‘V’ increases then the chances of overfitting also increases.
M is the number of base models.

Regularization [edit]

Fitting the training set too closely can lead to degradation of the model's generalization ability. Several so-called regularizat techniques reduce this overfitting effect by constraining the fitting procedure.

One natural regularization parameter is the number of gradient boosting iterations M (i.e. the number of trees in the model when the base learner is a decision tree). Increasing M reduces the error on training set, but setting it too high may lead to 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:

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

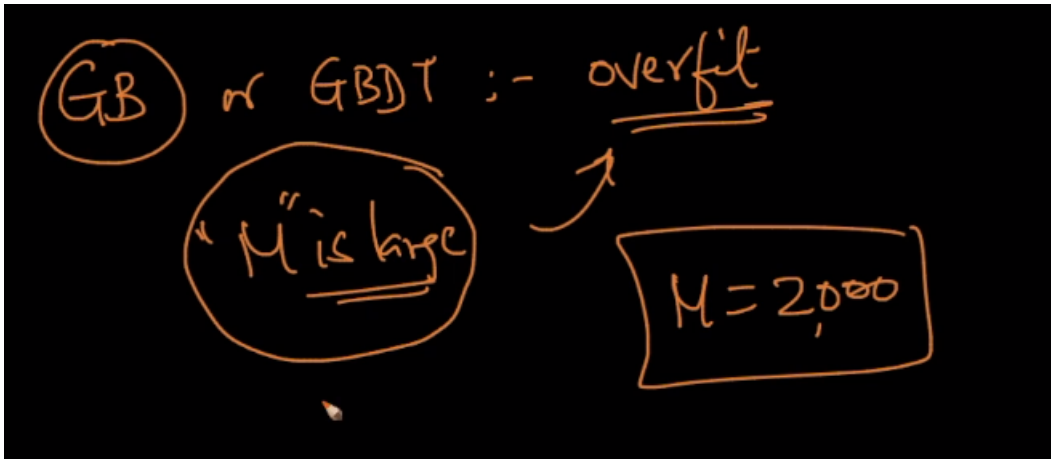
where parameter ν is called the "learning rate".

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.

Stochastic gradient boosting [edit]

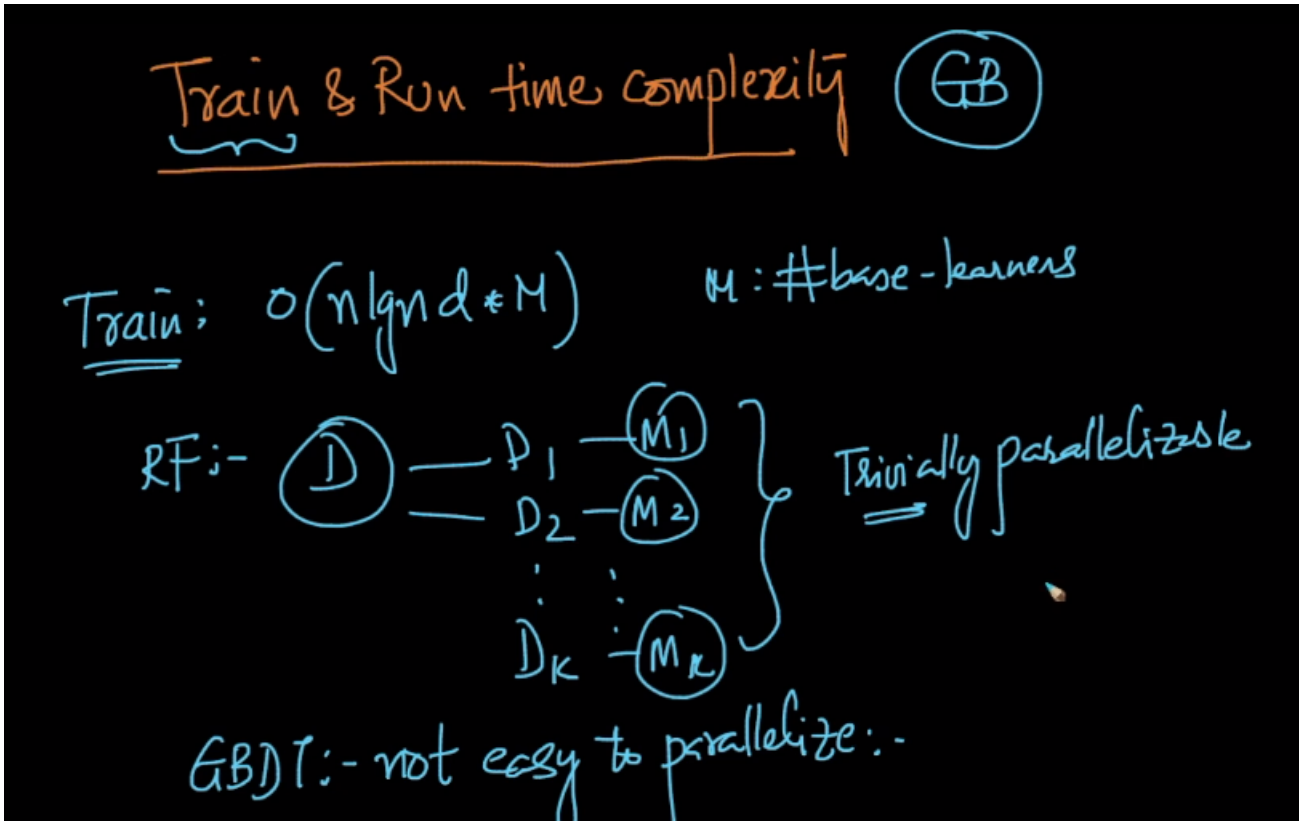
The ideal method is to find the best parameters using grid search.

It is easy to over – fit for the gradient boosted trees.

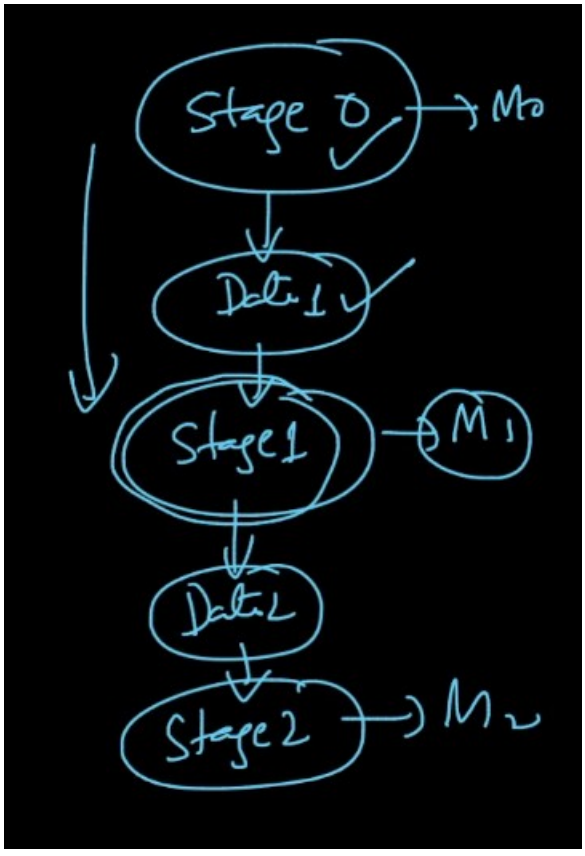


The train and test time complexity:

GBDT is not easy to parallelize.



In GBDT all the models are related to each other. The data is also produced after each of the stage.

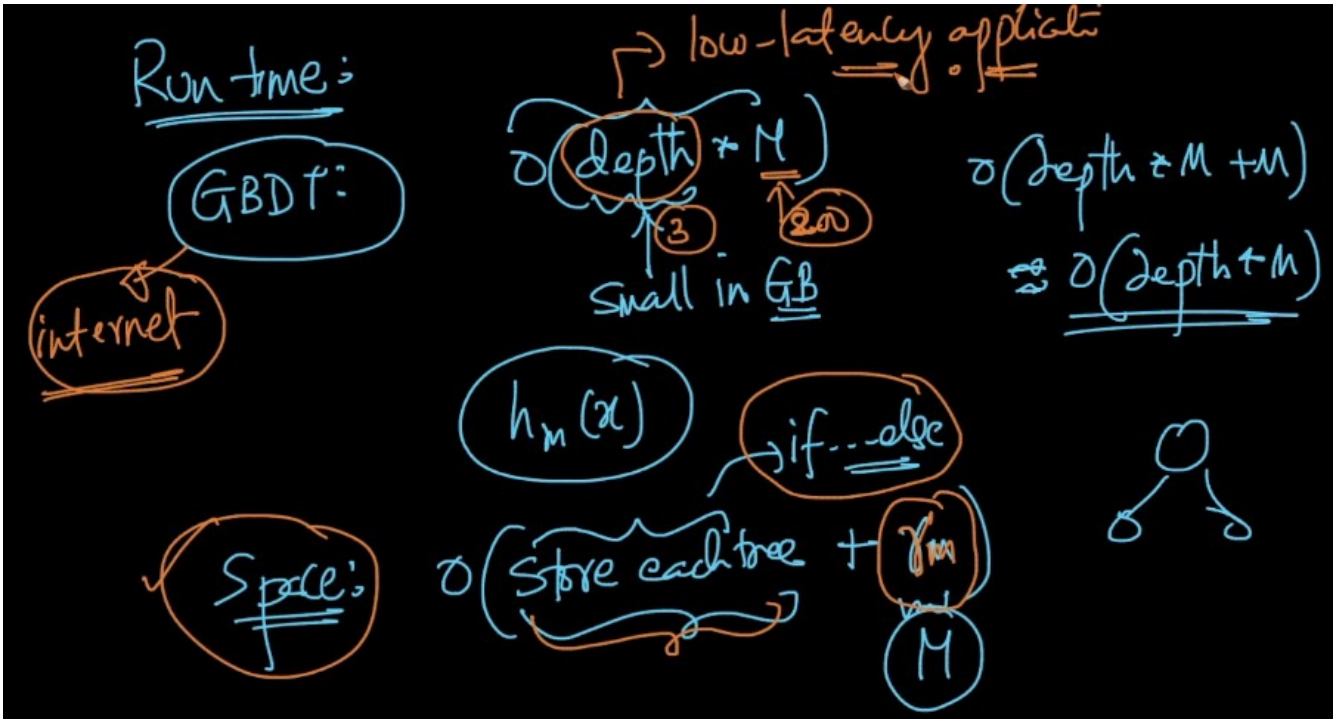


Hence GBDT is not trivial to parallalize.

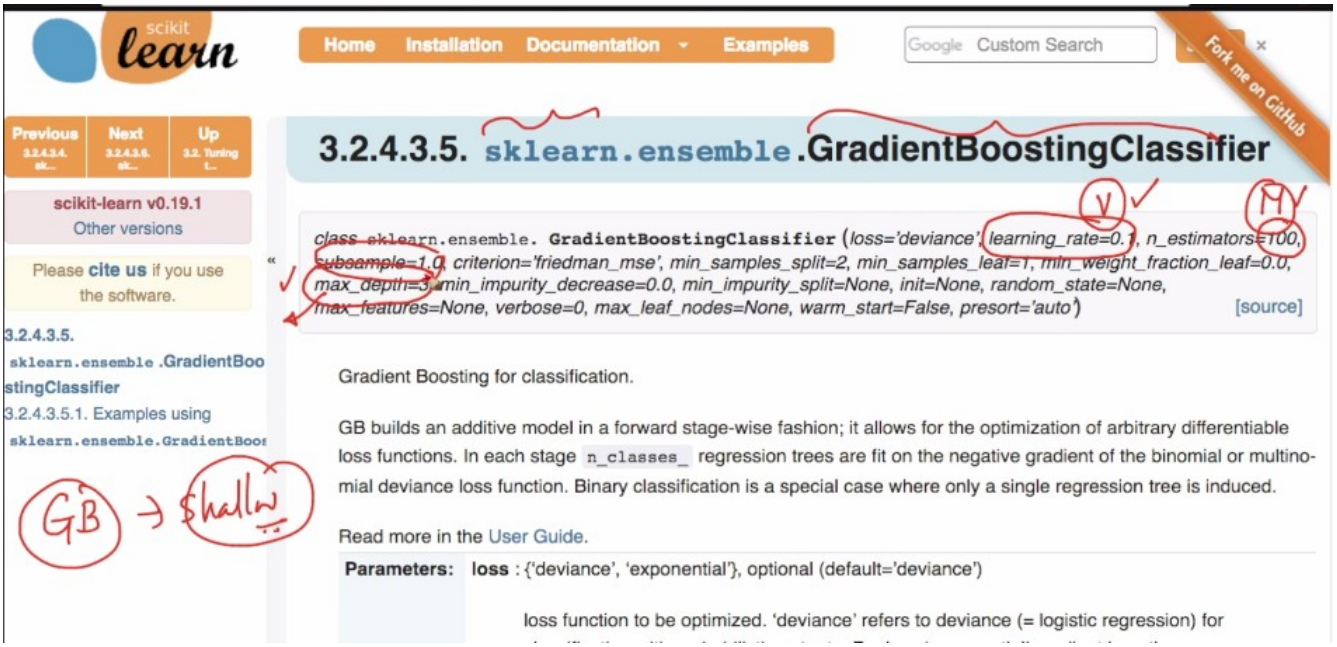
GBDT takes more time than random forest, even the time complexity remains same.

Run time:

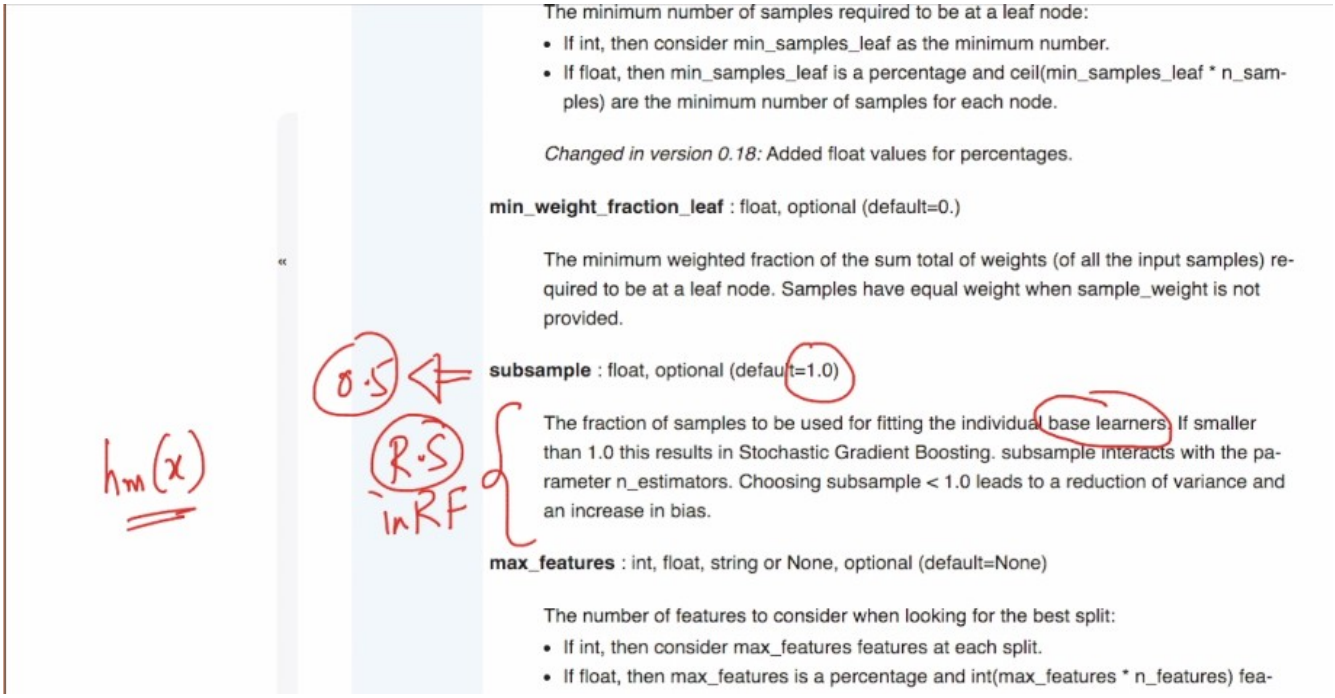
At run time the depth is very low and one must keep track of all the coefficient values.



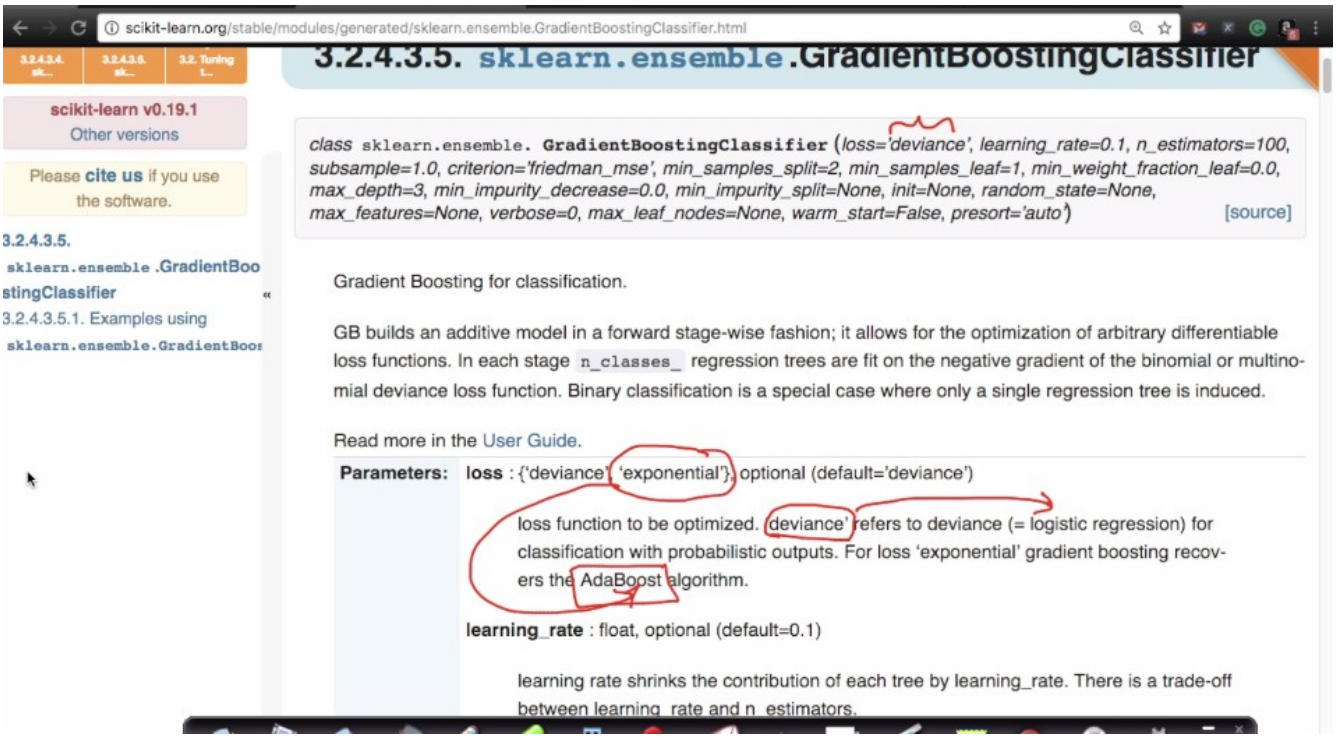
Gradient Boosting Classifier(Sklearn):



The sub sample can be taught of the row_sampling parameter.

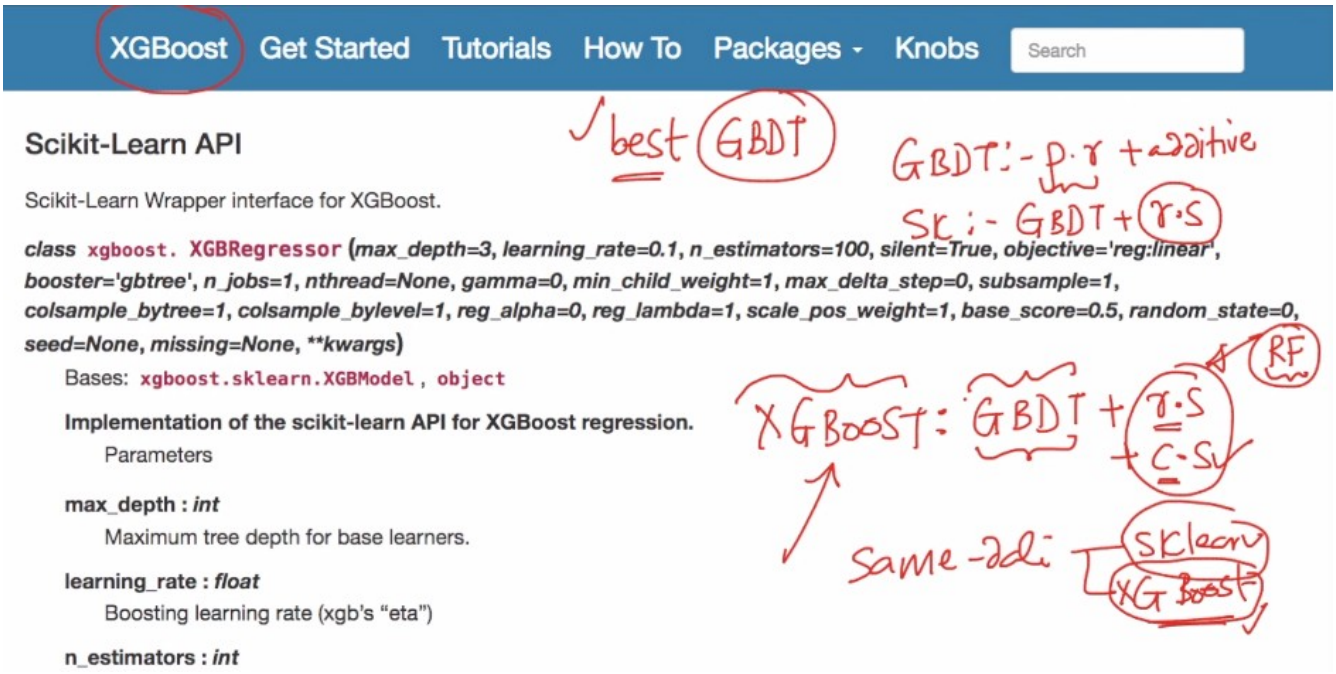


The value loss= , is the measure of type of loss function to be used.
If loss = ‘AdaBoost’, this algorithm is equal to another algorithm called ada – boost.

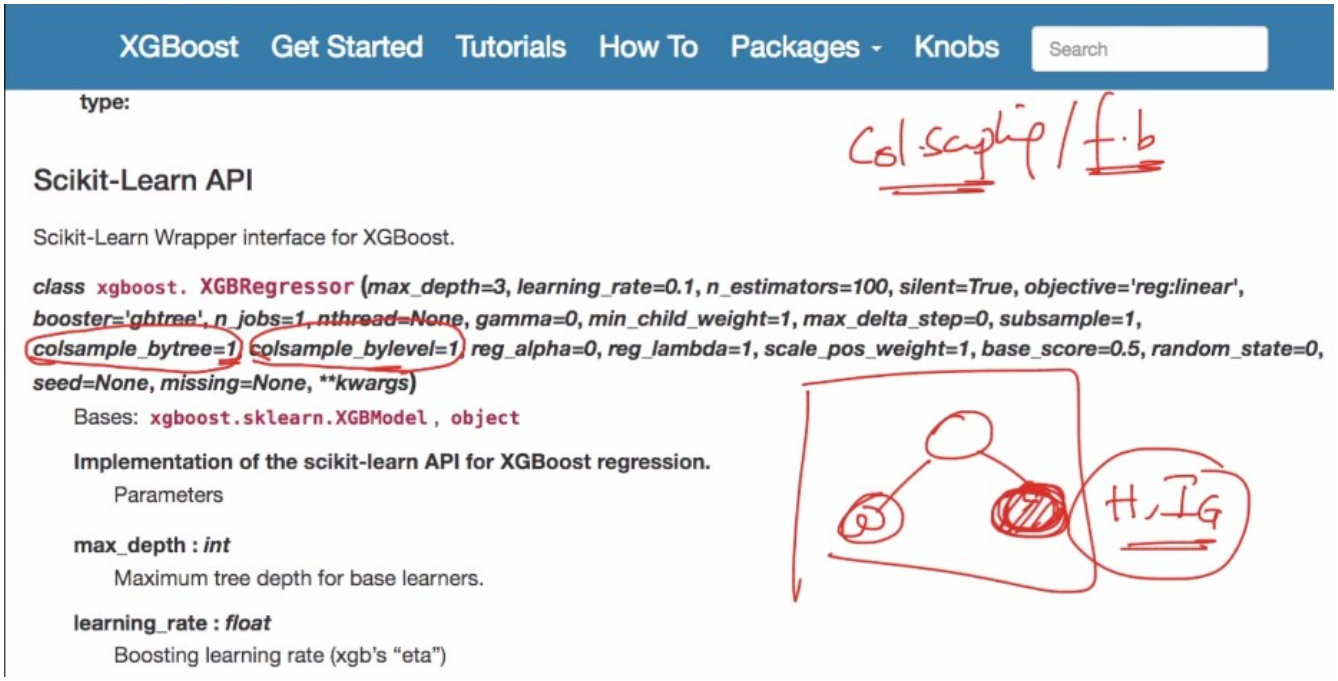


The sklearn implementation is not so efficient.
It is very slow.

Xgboost is the efficient way implementation.(well optimized).

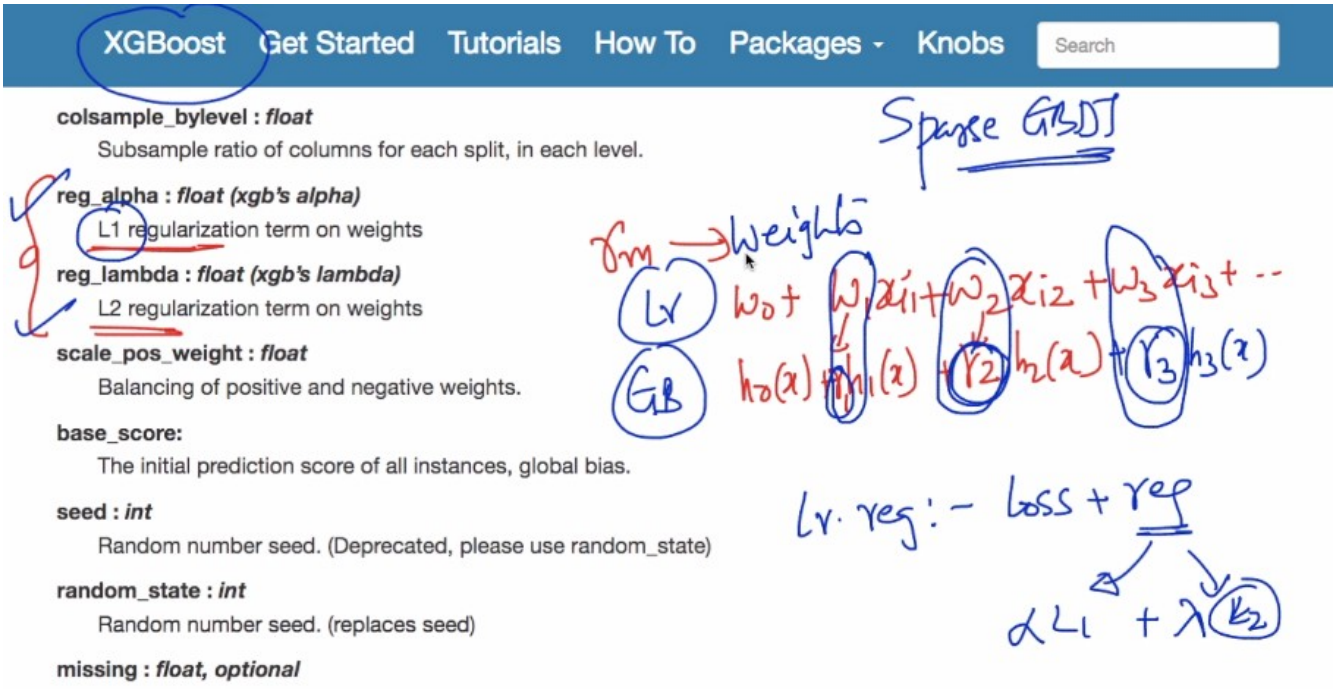


In XGBoost we use the GBDT + Row sampling + column sampling. XGBoost data sets perform well than the sklearn implementation due to several code optimizations.

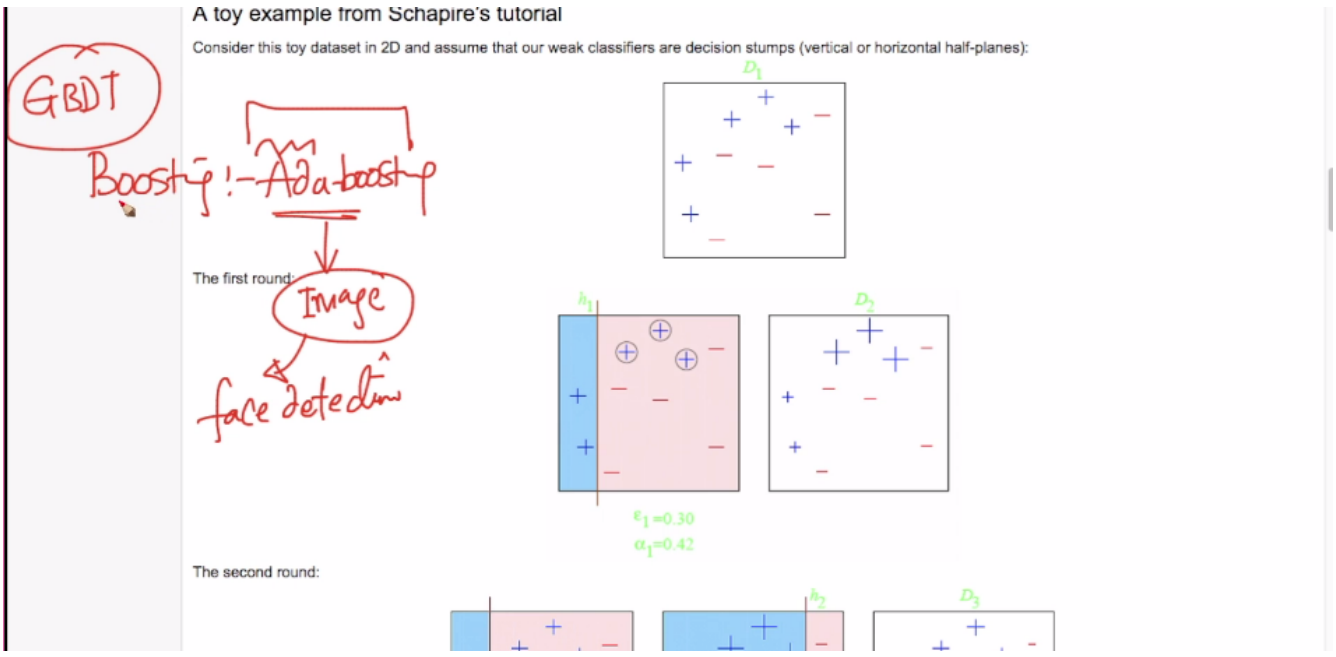


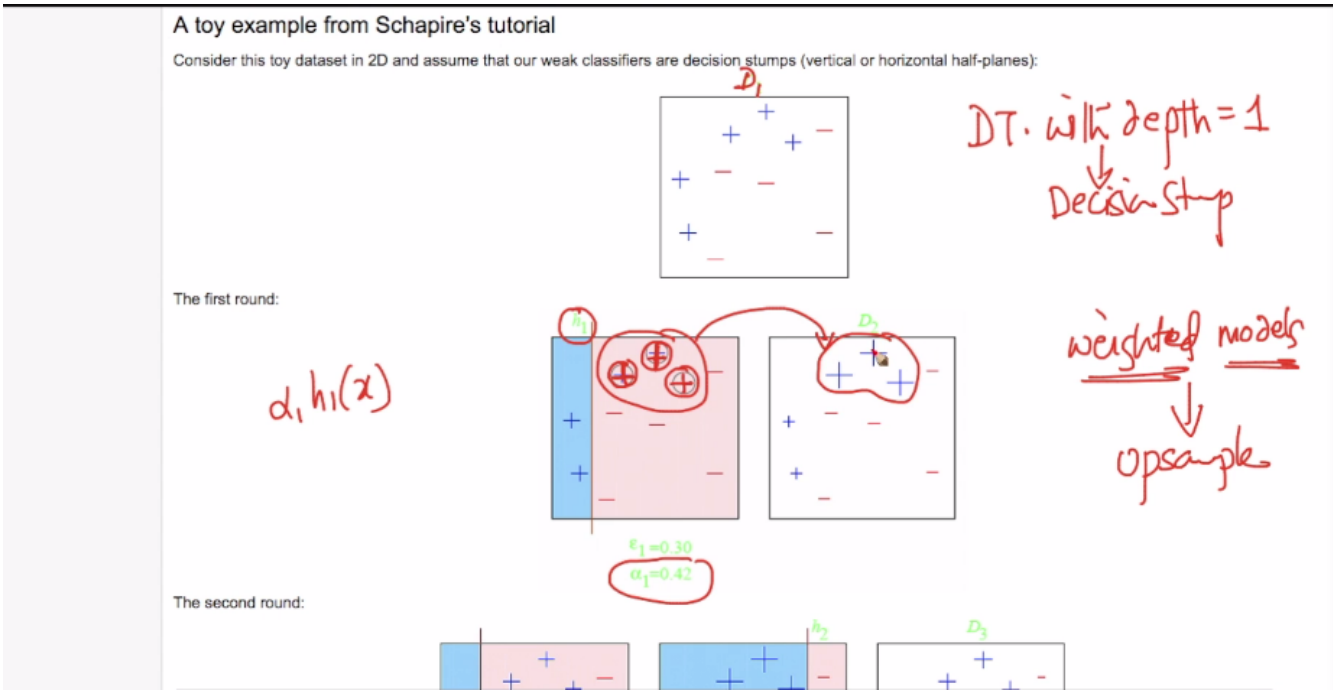
There is the choice of choosing the features at the level of tree and at the level each level in the tree.

There is the similar representation in case of linear regression and GB. We can look <gamma's> as the weights.



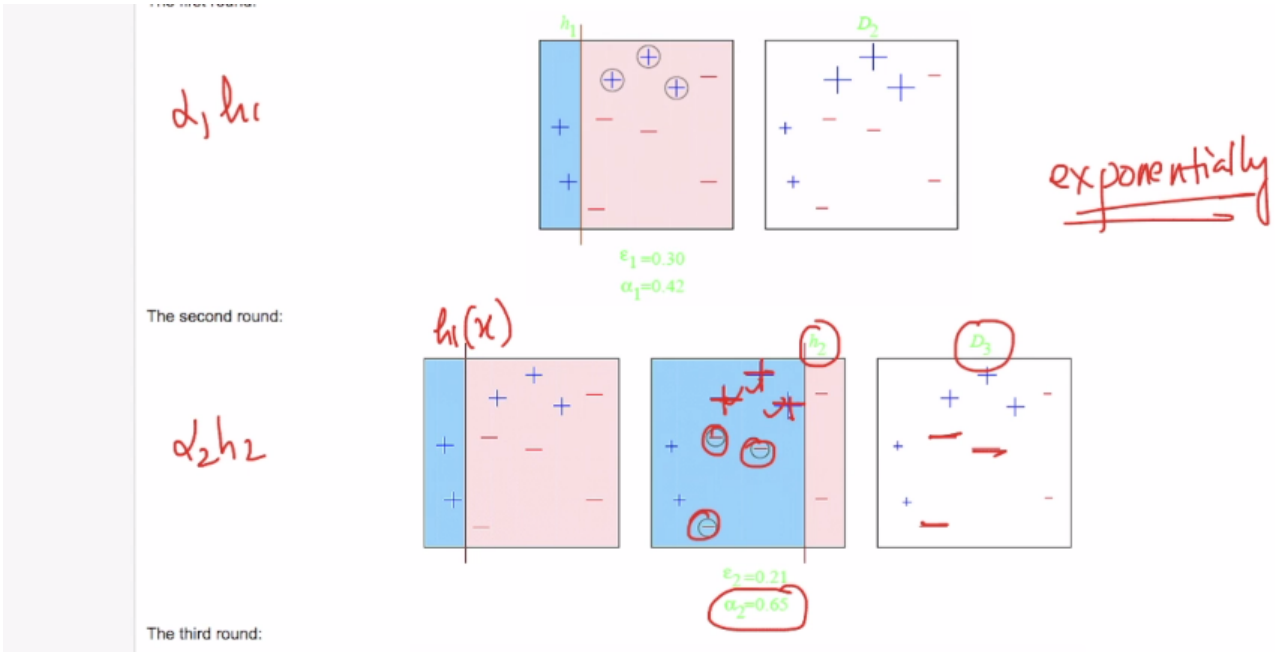
Ada-boost geometric intuition:
It is mainly used in case of image data for face detection.



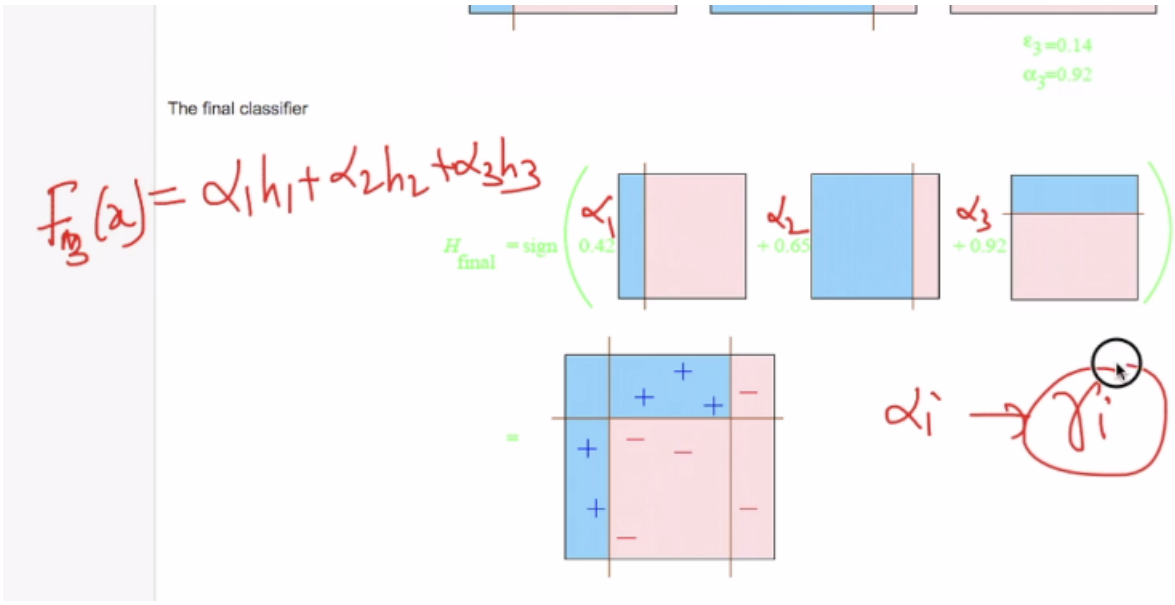


The error points in the data are up sampled in the next level.

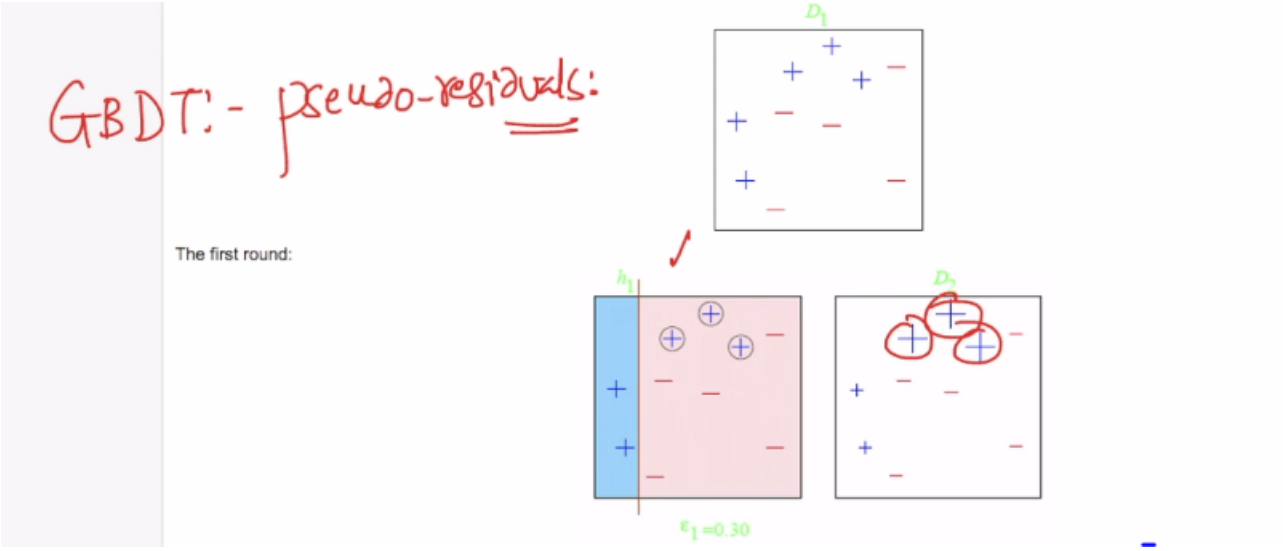
The weights are increased exponentially in adaptive boosting.
Even of the errors we also get the alpha2 as the weight for the error points.



In this way we implement more models with there weights.



In case of GBDT we use the pseudo – residuals.
In case if ADABOOST, we reduce the error-nous points.



The weights are increased exponentially and the miss-classified points are up sampled at each level.

Ada-boost is applied very less in real – world. It is mainly used for face detection.

Stacking models:

Mlxtend classifier for implementing stacking models.

In stacking the data is trained on several classification algorithms, the algos must be very different from each other. These are called the base models.(first – level classifiers). The more different the models are the better it works.

Then again construct the data set on the predictions made by the base models for every point in the data set.

Then on the new data set a classifier is trained to make the final output of the query point.

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

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

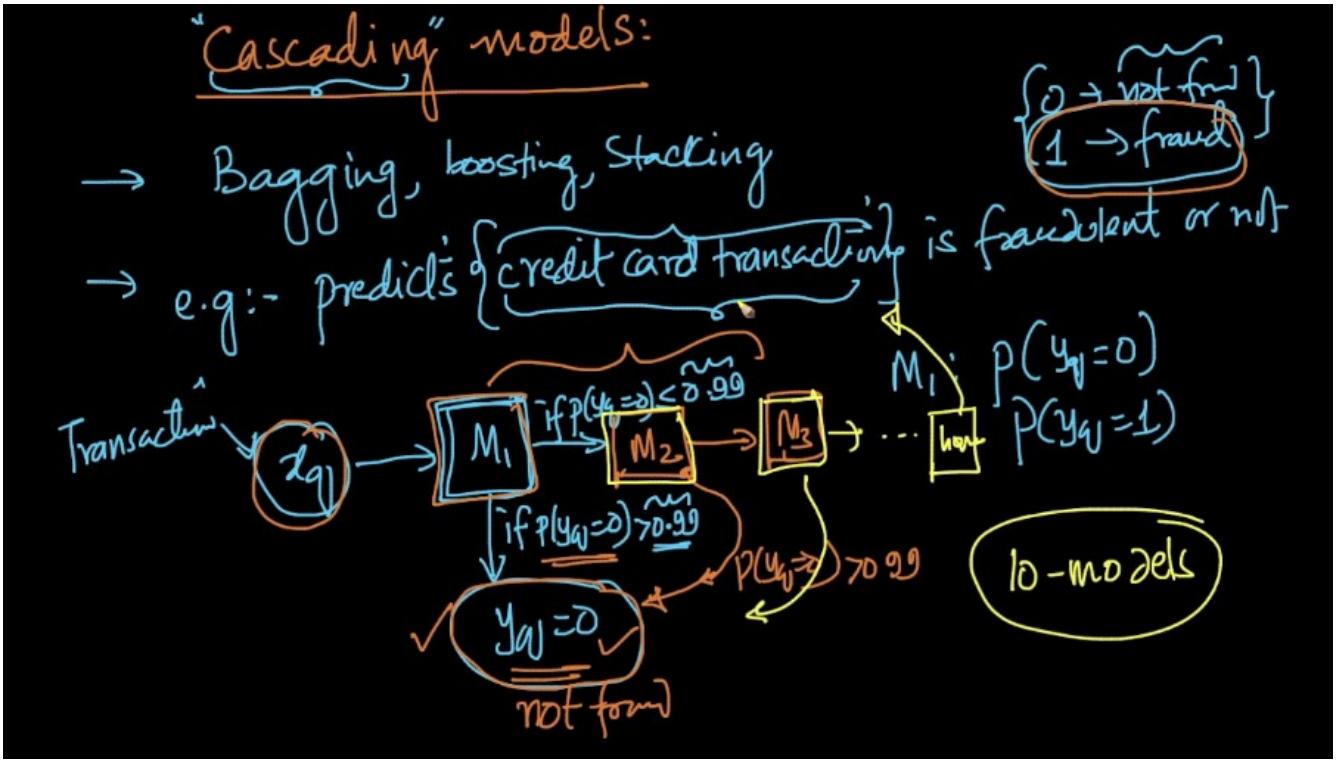
The last classifier is called the meta classifier.

Cascading models:

Example for understanding the cascading models. Every transaction is a vector, every fraud transaction can be done from various locations.

Every model can give the class probabilities. (fraud and not – fraud). If probability of the point to be one class can be large and other class is low.

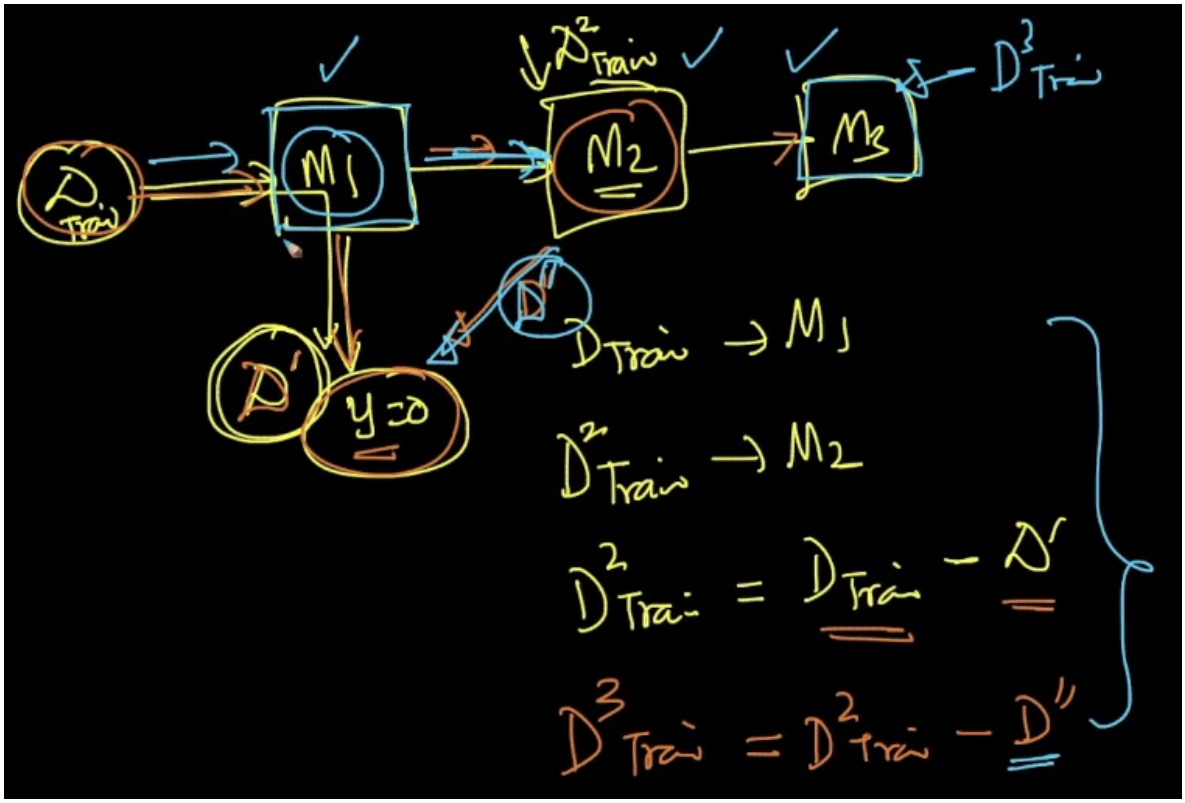
At every stage we train an another classifier to check the certainty of the model.



They are typically used when the cost of making mistake is very high.

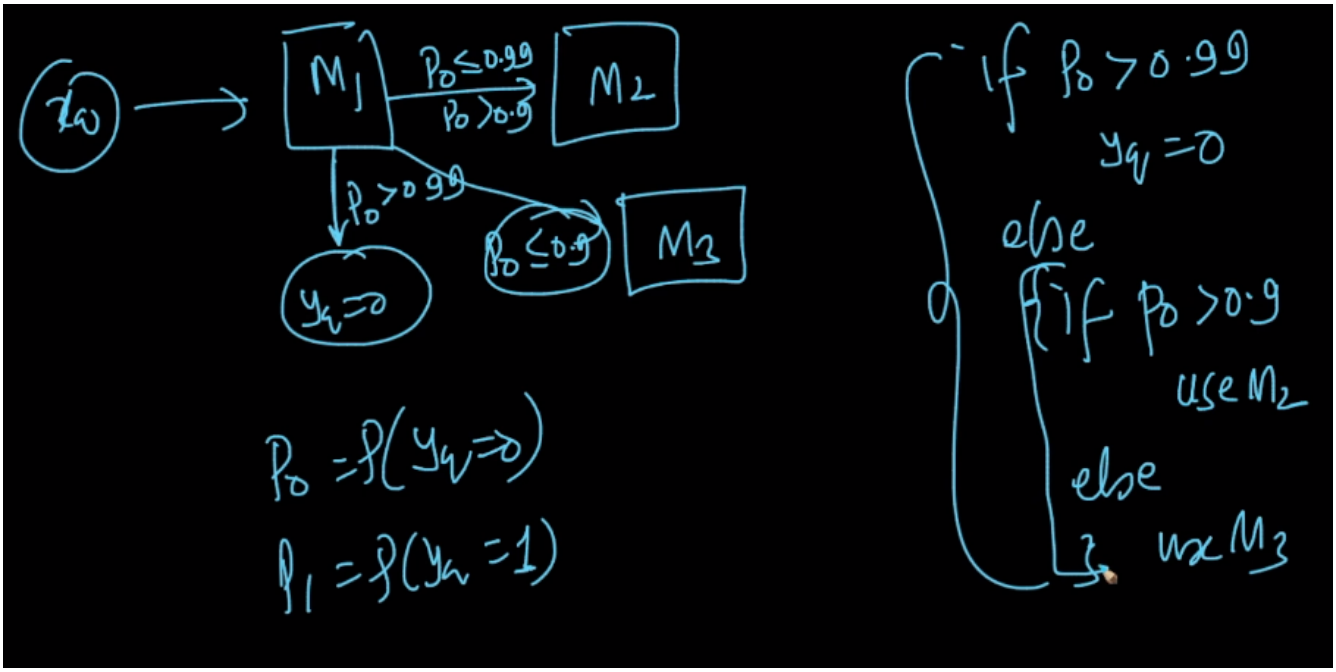
The points that don't give the accuracy that is being passed to the other model for achieving the accuracy and so on.. till there can be a specialist checking at the end.

At every level the data points in the data set are reduced.



The cascade can be more than one type.
The down output of the model is considered to be the class of the model based on the probability of the model.(i.e, say the model outputs 0.99 as the probability score, then the model will confirm the class of the data point, the remaining points are further sent to the cascade model of the first model).

We could make the complex if – else block statements for predicting the models outcome based on the accuracy that is needed.



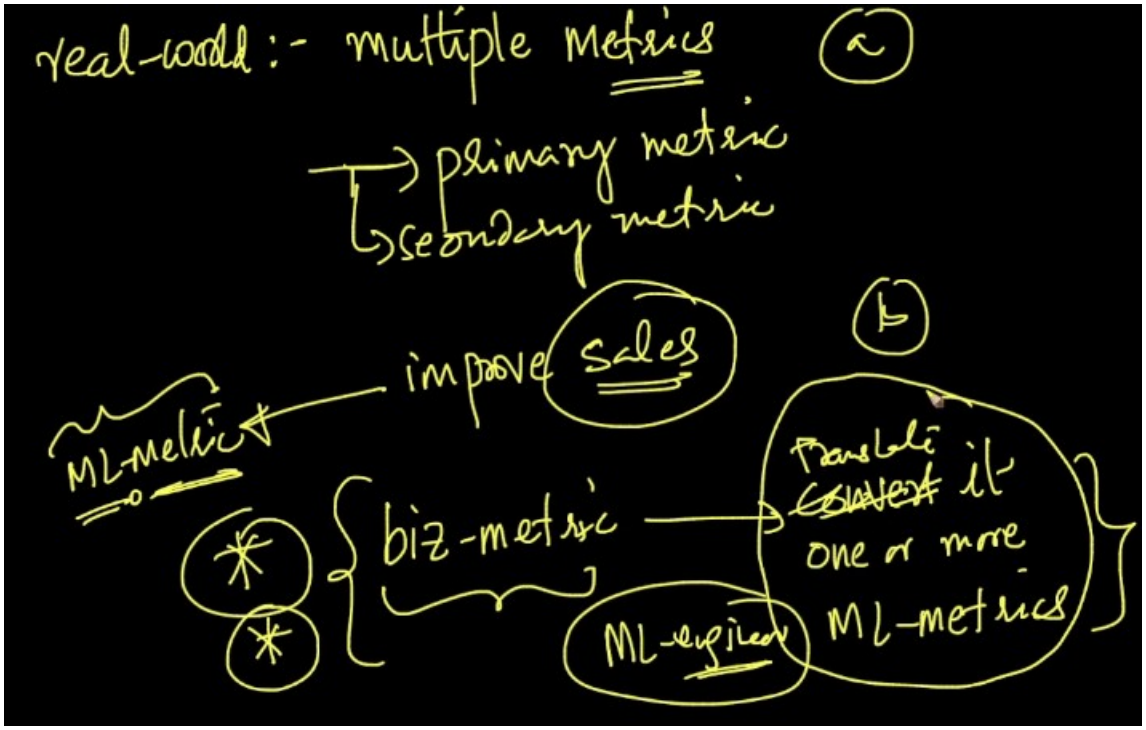
Cascades models can be extensively used in fraud detection, medicine applications for maintaining the accuracy of the model.

They can also be build up to 20 levels.

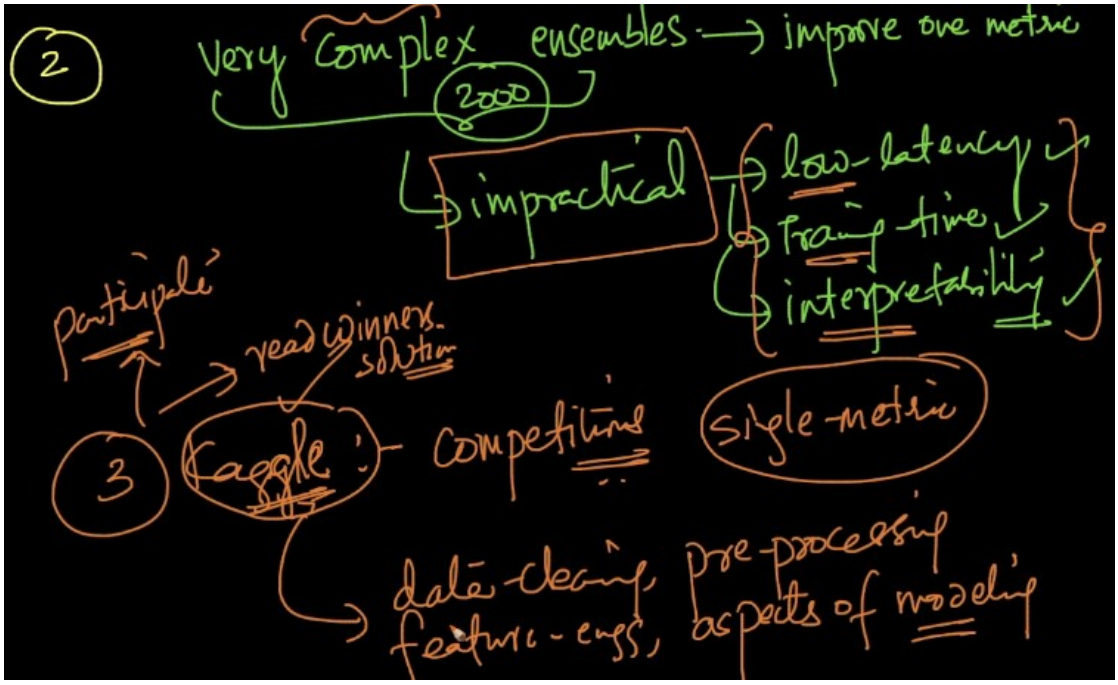
If one transaction is lost then there will be huge loss.

Kaggle vs. Real – world cases:

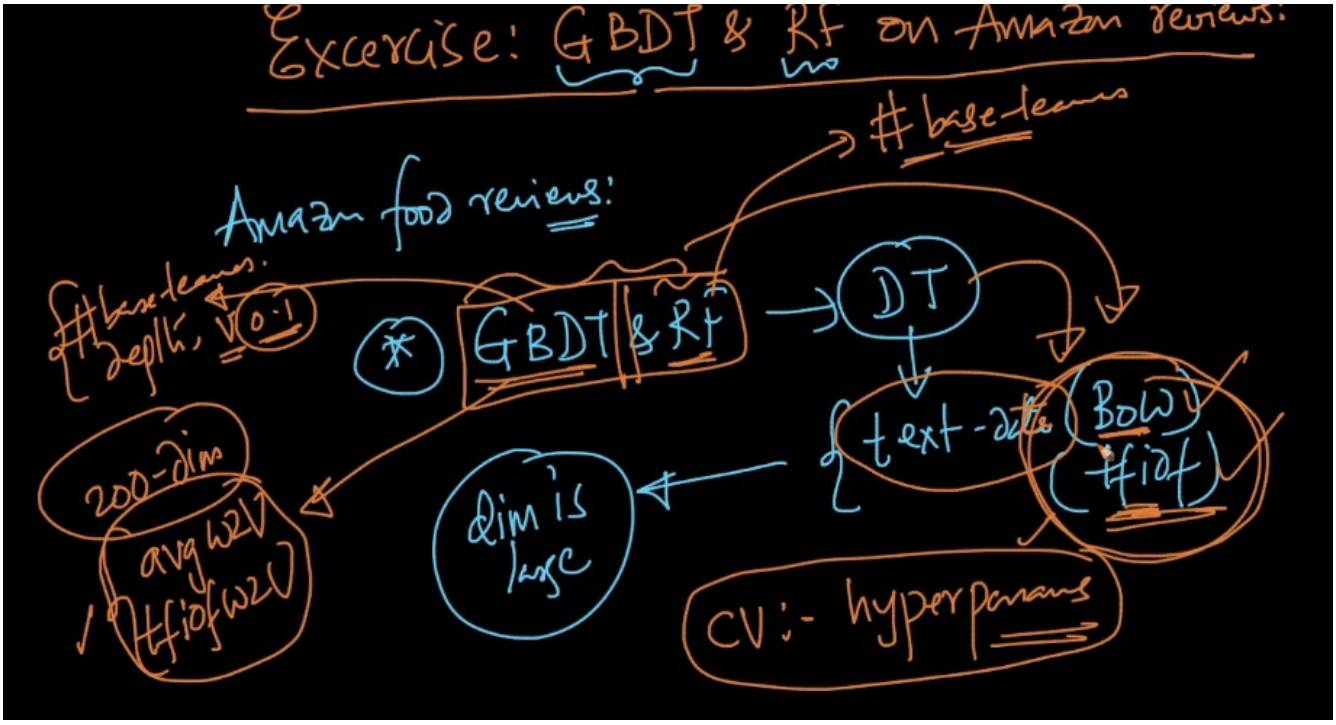
Kaggle can only take one metric as the measure. A real world there are multiple metrics that can be taken care of that favors the business problem. They take the business goal and translate the biz-problem to one – more machine learning metrics.



In case – studies we will see the translation of the biz – problem to a machine learning metrics. They build the very complex ensembles, this may be impractical.



Exercise:


Excercise: GBDT & RF on Amazon reviews!
Amazon food reviews!
#base-learners
GBDT & RF
DT
text-data
BoW
tfidf
200-dims
avg w2v
tfidf w2v
dim is huge
CV:- hyperparams