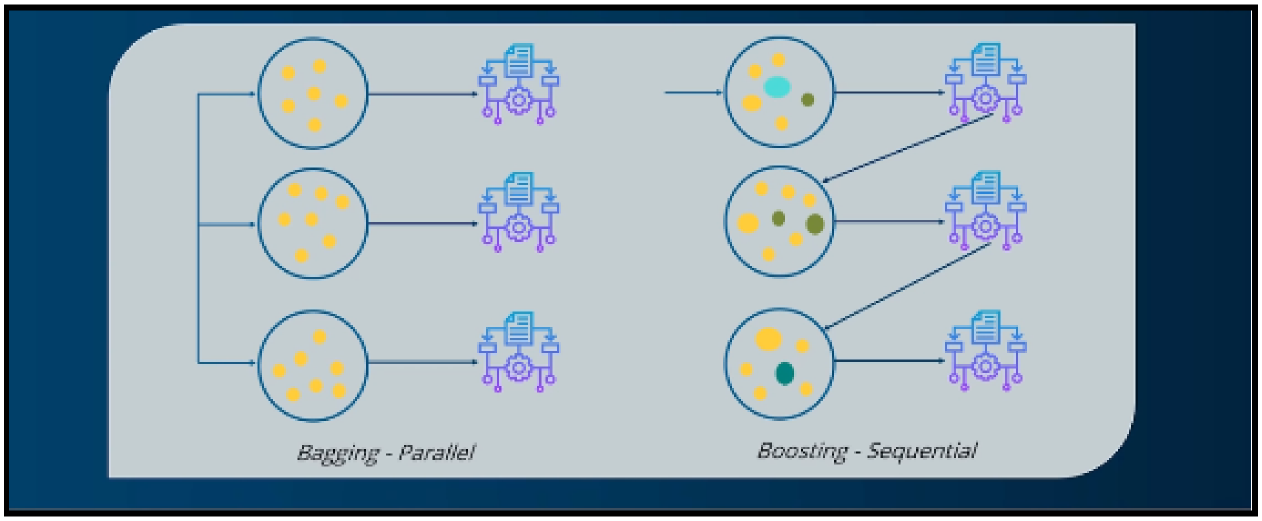
Machine Learning technique - Boosting

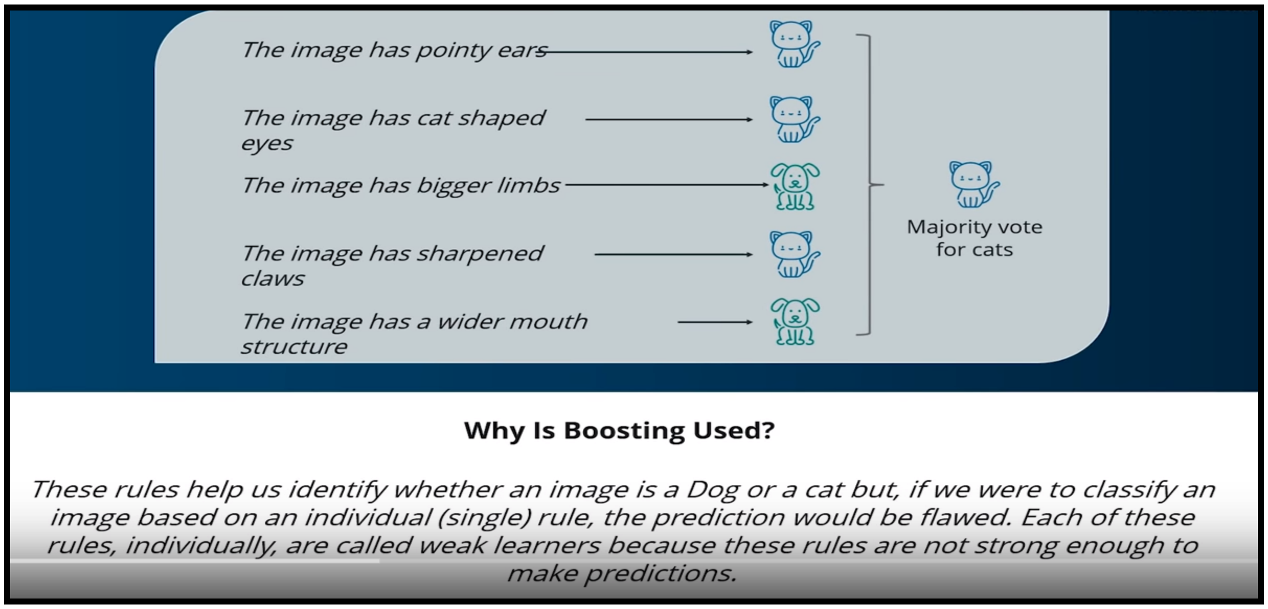
Boosting technique is ensemble learner which takes only one model either linear or decision tree, whereas by default it uses the decision tree.

What is Ensemble: Ensemble learning offers a systematic solution to combine the predictive power of multiple learners. The resultant is a single model which gives the aggregated output from several models.

Image: 1: Workflow diagram for bagging and boosting ensemble

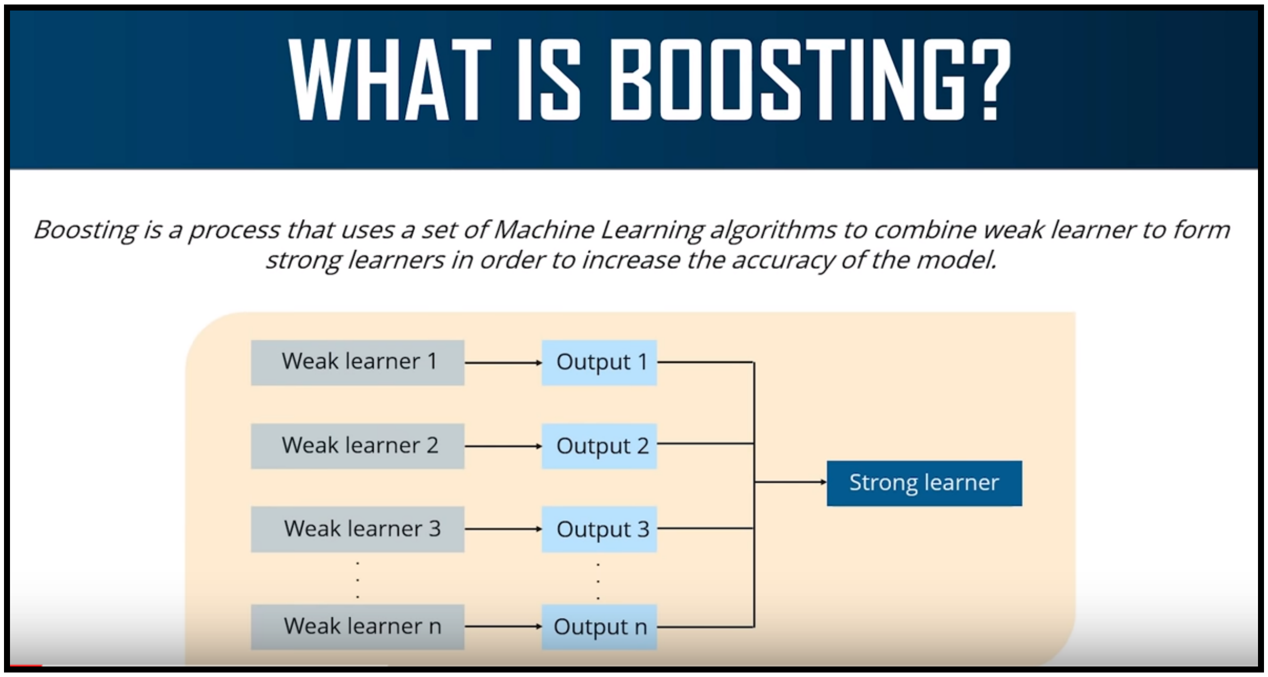


Why we need a Boosting technique - Let's take the example of classifying cat and dog image.


The image has pointy ears
The image has cat shaped eyes
The image has bigger limbs
The image has sharpened claws
The image has a wider mouth structure
Majority vote for cats

Why Is Boosting Used?

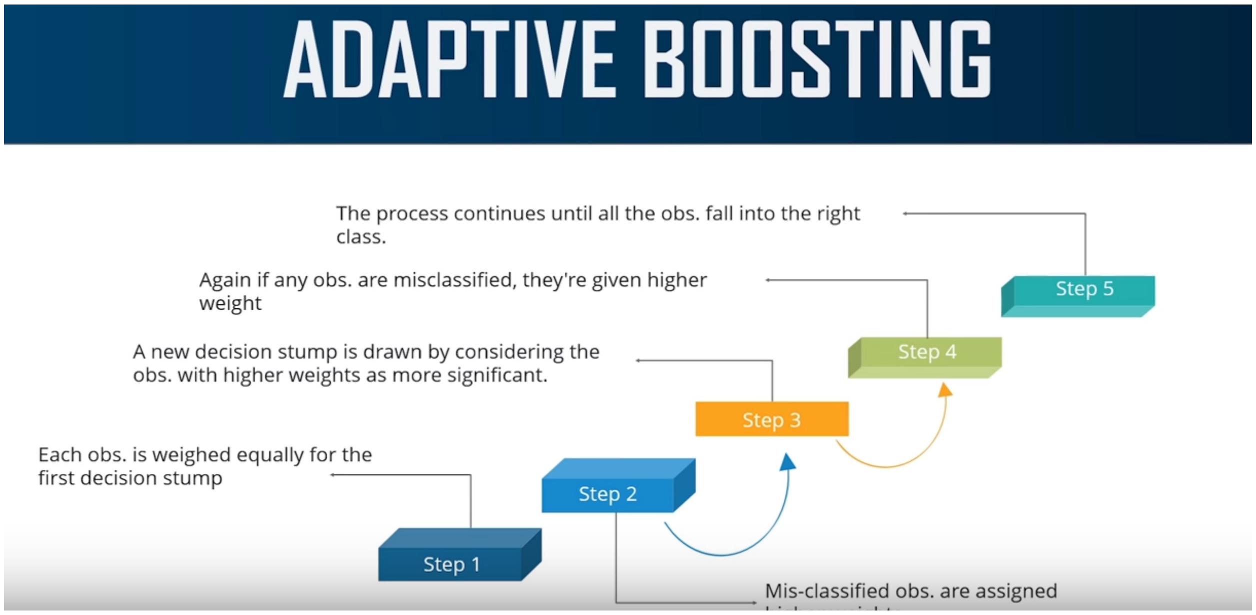
These rules help us identify whether an image is a Dog or a cat but, if we were to classify an image based on an individual (single) rule, the prediction would be flawed. Each of these rules, individually, are called weak learners because these rules are not strong enough to make predictions.



WHAT IS BOOSTING?

Boosting is a process that uses a set of Machine Learning algorithms to combine weak learner to form strong learners in order to increase the accuracy of the model.

If you see Image1: It's showing the way boosting method works. Every time we pass the complete data set to a weak learner (model) and for whichever records it's not able to predict the output correctly we give more weightage to that particular record so that in our next weak learner that particular records get predicted perfectly.



ADAPTIVE BOOSTING

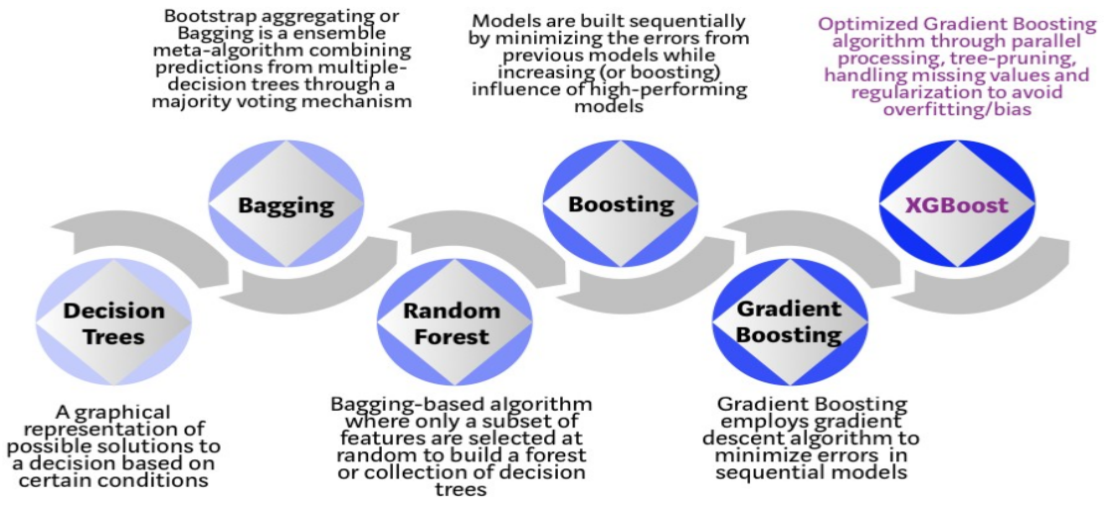
The process continues until all the obs. fall into the right class.
Again if any obs. are misclassified, they're given higher weight
A new decision stump is drawn by considering the obs. with higher weights as more significant.
Each obs. is weighed equally for the first decision stump
Step 5
Step 4
Step 3
Step 2
Step 1
Mis-classified obs. are assigned

Step2: Misclassified obs. are assigned higher weightage

# **Gradient Boosting**: It is almost similar to Adaboost but here we don't do weight updation for misclassified records. Instead of weight updation, we optimize the loss function of the previous model. (takes longer time to compute)

Components used- 1. Loss function
2. Weak learner (model)
3. The additive model that regularize the loss function (new model)

# **XG BOOST**: It's the updated version of gradient boosting which focuses on computational power and model efficiency.

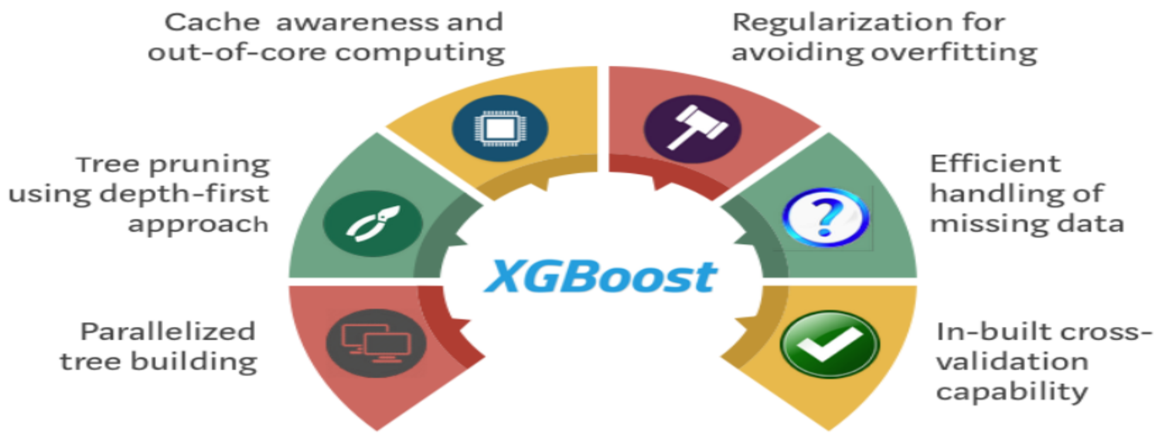


Bootstrap aggregating or Bagging is a ensemble meta-algorithm combining predictions from multiple-decision trees through a majority voting mechanism
Models are built sequentially by minimizing the errors from previous models while increasing (or boosting) influence of high-performing models
Optimized Gradient Boosting algorithm through parallel processing, tree-pruning, handling missing values and regularization to avoid overfitting/bias
Bagging
Boosting
XGBoost
Decision Trees
Random Forest
Gradient Boosting
A graphical representation of possible solutions to a decision based on certain conditions
Bagging-based algorithm where only a subset of features are selected at random to build a forest or collection of decision trees
Gradient Boosting employs gradient descent algorithm to minimize errors in sequential models

Evolution of XGBoost Algorithm from Decision Trees

Key features -

- Parallelization (create many trees parallelly)
- Distributed computing (helps to evaluate quickly)
- Out-of Core computing (able to handle varied data set)
- Cache optimization (enables to use best possible of your hardware configuration)

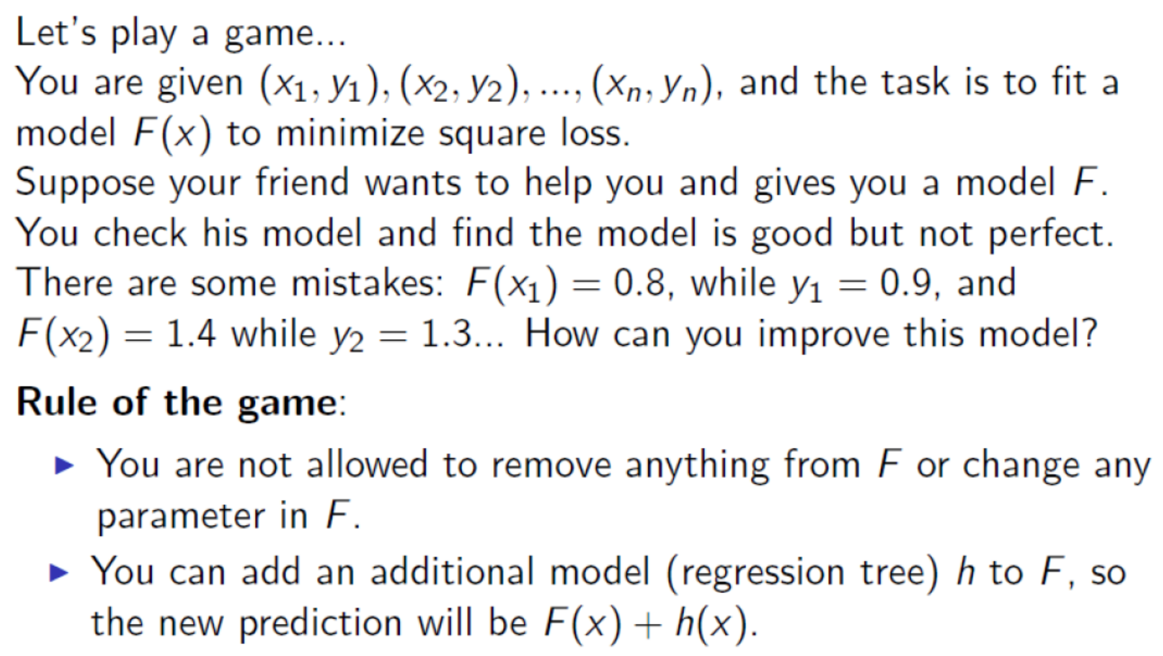


Cache awareness and out-of-core computing
Regularization for avoiding overfitting
Tree pruning using depth-first approach
Efficient handling of missing data
XGBoost
Parallelized tree building
In-built cross-validation capability

How XGBoost optimizes standard GBM algorithm

Maths behind GBM or XG Boost:-

Let’s understand it using game



Let’s play a game...
You are given $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, and the task is to fit a model $F(x)$ to minimize square loss.
Suppose your friend wants to help you and gives you a model F.
You check his model and find the model is good but not perfect.
There are some mistakes: $F(x_1) = 0.8$, while $y_1 = 0.9$, and $F(x_2) = 1.4$ while $y_2 = 1.3$... How can you improve this model?

Rule of the game:

- You are not allowed to remove anything from F or change any parameter in F.
- You can add an additional model (regression tree) h to F, so the new prediction will be $F(x) + h(x)$.

Simple solution:

You wish to improve the model such that

$$\begin{aligned} &F(x_1) + h(x_1) = y_1 \\ &F(x_2) + h(x_2) = y_2 \\ &\ldots \\ &F(x_n) + h(x_n) = y_n \end{aligned}$$

Or, equivalently, you wish

$$\begin{aligned} &h(x_1) = y_1 - F(x_1) \\ &h(x_2) = y_2 - F(x_2) \\ &\ldots \\ &h(x_n) = y_n - F(x_n) \end{aligned}$$

Just fit a regression tree h to data
$(x_1, y_1 - F(x_1)), (x_2, y_2 - F(x_2)), \ldots, (x_n, y_n - F(x_n))$

$y_i - F(x_i)$ are called **residuals**. These are the parts that existing model F cannot do well.
The role of h is to compensate the shortcoming of existing model F.

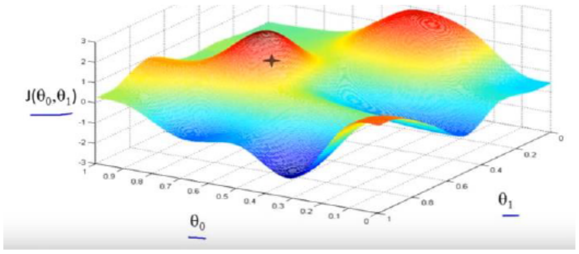
If the new model $F + h$ is still not satisfactory, we can add another regression tree...

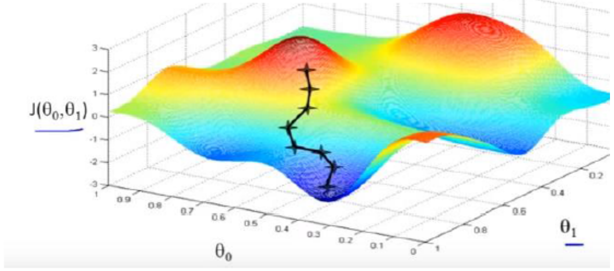
How is this related to gradient descent?

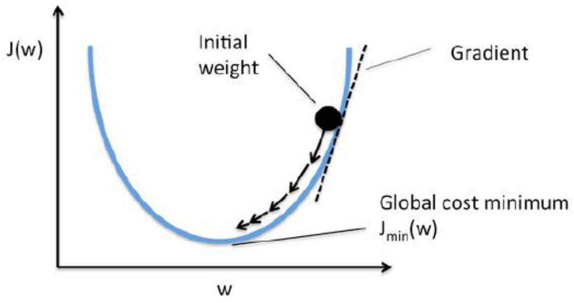
Gradient Descent :

Minimize a function by moving in the opposite direction of the gradient.

$$\theta_i := \theta_i - \rho \frac{\partial J}{\partial \theta_i}$$







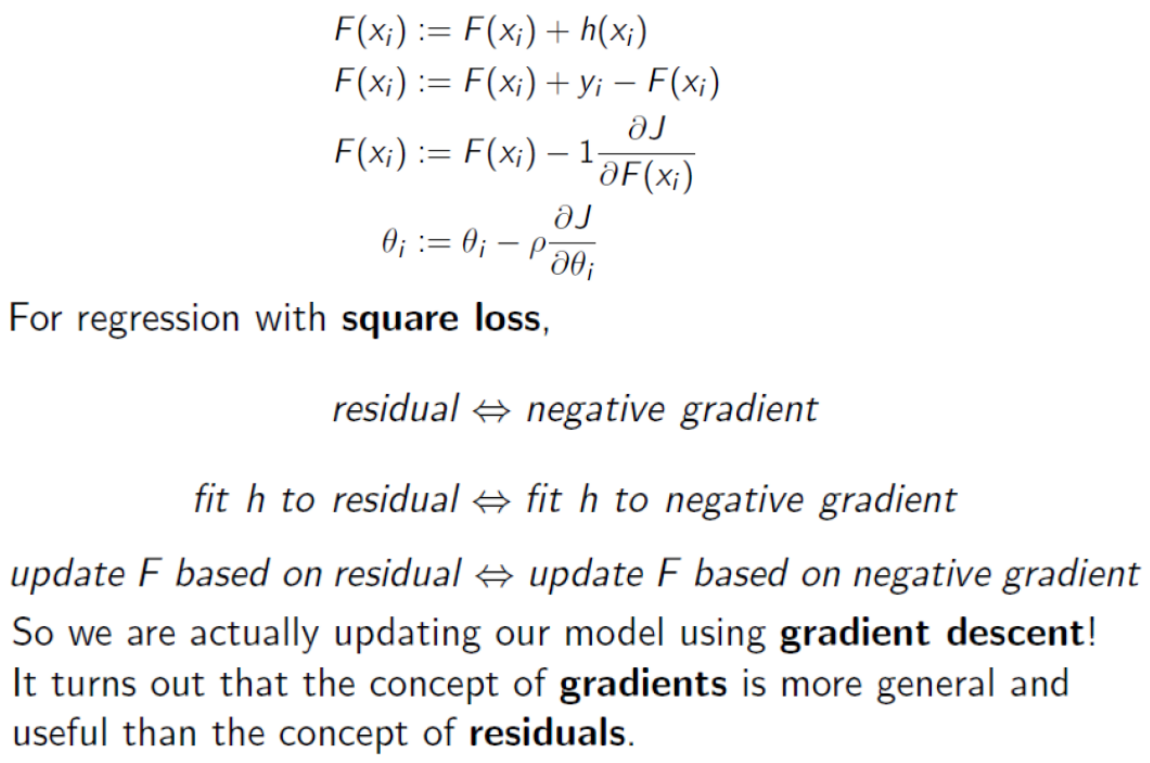
Loss function $L(y, F(x)) = (y - F(x))^2/2$
We want to minimize $J = \sum_i L(y_i, F(x_i))$ by adjusting $F(x_1), F(x_2), ..., F(x_n)$.

$$\frac{\partial J}{\partial F(x_i)} = \frac{\partial \sum_i L(y_i, F(x_i))}{\partial F(x_i)} = \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = F(x_i) - y_i$$

So we can interpret residuals as negative gradients.

$$y_i - F(x_i) = -\frac{\partial J}{\partial F(x_i)}$$

Relationship between Gradient descent and GBM:



$$F(x_i) := F(x_i) + h(x_i)$$
$$F(x_i) := F(x_i) + y_i - F(x_i)$$
$$F(x_i) := F(x_i) - 1\frac{\partial J}{\partial F(x_i)}$$
$$\theta_i := \theta_i - \rho\frac{\partial J}{\partial \theta_i}$$

For regression with **square loss**,

$$\textit{residual} \Leftrightarrow \textit{negative gradient}$$

$$\textit{fit h to residual} \Leftrightarrow \textit{fit h to negative gradient}$$

$$\textit{update F based on residual} \Leftrightarrow \textit{update F based on negative gradient}$$

So we are actually updating our model using **gradient descent**!
It turns out that the concept of **gradients** is more general and useful than the concept of **residuals**.