EMMA ding

emmading.com
info@datainterviewpro.com

Gradient Boosting

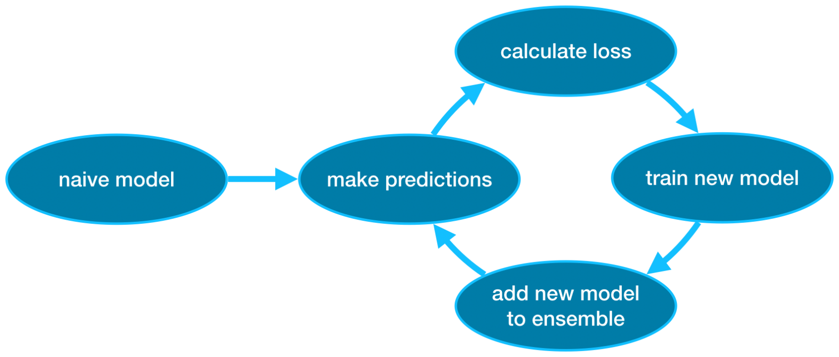
Lesson Structure

📌 Interview Questions

- What is gradient boosting method?
- Describe the architecture of gradient boosting
- Which of the following are appropriate methods of addressing high variance in a Gradient Boosting model?
  - Increase the number of trees
  - Use L1 or L2 regularization
  - Use randomly selected sub-samples
  - None of the above
- What is XGBoost?

▼ Gradient Boosting

- a.k.a Gradient Boosting Machines (GBMs)
- An ensemble learning algorithm which is widely used in industrial applications and machine learning competitions.
- A **supervised learning** algorithm, which attempts to accurately predict a target by combining the estimates of a set of simpler and weaker learners.
  - Learners learn sequentially
  - Convert many weak learners into a complex learner
- It's called gradient boosting because it uses a **gradient descent** procedure to minimize the loss when adding new learners to the ensemble.



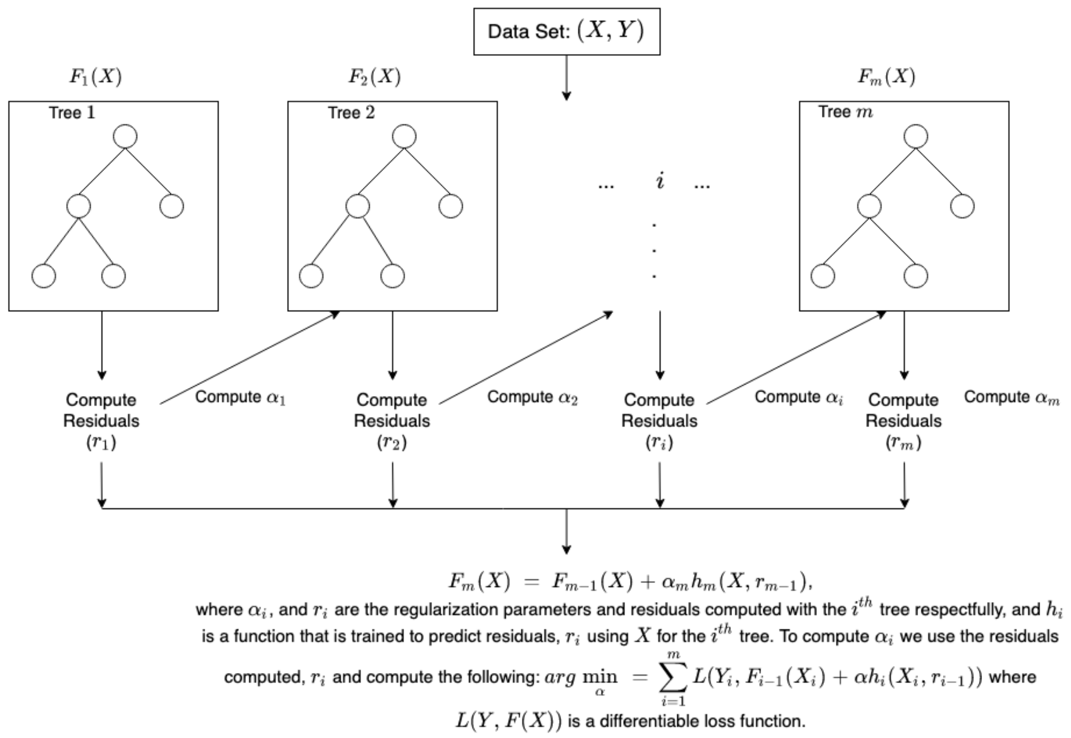
Source: https://www.kaggle.com/code/alexisbcook/xgboost

▼ Gradient-boosted Trees

- Weak learner is Classification and Regression Trees (CART)
- Builds decision trees in an **iterative** fashion using prediction residuals (the difference between the target and the predicted value)
    - In each round, a new tree is fit on the **residuals** of the previous tree.
    - The model improves as we are moving each tree more in the right direction via small updates. These updates are based on a **loss gradient**.
    - The training proceeds **iteratively**, adding new trees that predict the residuals of previous trees that are then combined with previous trees to make the final prediction.

▼ Algorithm

1. Start with a simple model to return a constant value**.**
    - Use a decision tree root node (e.g. a tree with a single leaf note) $F_0(X)$
2. For each tree $i = 1, ..., m$, where m is the predefined maximum number of trees.



$$F_m(X) = F_{m-1}(X) + \alpha_m h_m(X, r_{m-1}),$$

where α_i, and r_i are the regularization parameters and residuals computed with the i^{th} tree respectfully, and h_i is a function that is trained to predict residuals, r_i using X for the i^{th} tree. To compute α_i we use the residuals computed, r_i and compute the following: $arg \min_{\alpha} = \sum_{i=1}^{m} L(Y_i, F_{i-1}(X_i) + \alpha h_i(X_i, r_{i-1}))$ where $L(Y, F(X))$ is a differentiable loss function.

How gradient boosted tree works. https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost-HowItWorks.html

▼ Compute residual r_i

- r_i is the negative gradient of the loss function with respect to the prediction of the previous tree

$$r_i = -[\frac{\partial L(Y, F(X))}{\partial F(X)}]_{F(X)=F_{i-1}(X)}$$

- $L(Y, F(X))$ is a differentiable loss function.
    - For MSE: $L(Y, F(X)) = \sum(Y - F(X))^2$
- $F(X)$ is the prediction of the previous tree $F_{i-1}(X)$.

▼ Fit a new tree to predict r_i using all features

Subsequent learners are trained to predict errors of the previous prediction.

$h_i(X, r_{i-1}) \approx \nabla r_i$

- $h_i(X, r_{i-1})$ is a function to predict r_i.
- ∇r_i: gradient of r_i with respect to the prediction $F(X)$.
    - for MSE: $\nabla r_i = Y - F(X)$

▼ Update the prediction

$F_i(X) = F_{i-1}(X) + \alpha h_i(X, r_{i-1})$

- $F_{i-1}(X)$ is the prediction of the previous tree.
- α is the learning rate, typically a small value between 0.01 and 1.
- We scale h_i by α to update the model incrementally by taking small steps, which helps avoid overfitting.

3. Output $F_m(X)$.
   - Overall prediction given by a weighted sum of the collection

▼ Hyperparameters

Boosting reduces bias and increases variance by increasing the complexity of weak learners. It can overfit. By tuning the hyperparameters, overfitting can be prevented.

- Number of trees m (i.e. number of iterations) - increasing m reduces the error on training set (bias), but setting it too high may lead to overfitting.
- Max depth of trees - increasing the max depth will make the model more complex and more likely to overfit.
- Learning rate α - small learning rates (< 0.1) yield dramatic improvements in models' generalization ability with increasing computational time (both during training and prediction).
- Subsample size (randomly sample a fraction f of the size of the training data prior to growing trees) - smaller values of f introduce randomness into the algorithm and help prevent overfitting.

📌 Which of the following are appropriate methods of addressing high variance in a Gradient Boosting model? (Select all that apply)
- Increase the number of trees
✔️ Use L1 or L2 regularization
✔️ Use randomly selected sub-samples
- None of the above

▼ Pros and Cons

- Pros
  - It produces very accurate models, it outperforms random forest in accuracy.
  - No data pre-processing required - often works well with categorical and numerical values as is.
  - Handles missing data - imputation not required.
- Cons
  - Gradient boosting is a sequential process that can be slow to train.
  - Computationally expensive - often require many trees (>1000) which can be time and memory exhaustive.
  - Sacrifices interpretability for accuracy - less interpretative in nature.

        - e.g., it is self-explained to follow the path that a decision tree takes to make predictions but following the paths of thousands of trees in gradient-boosted trees is much harder.

▼ XGBoost

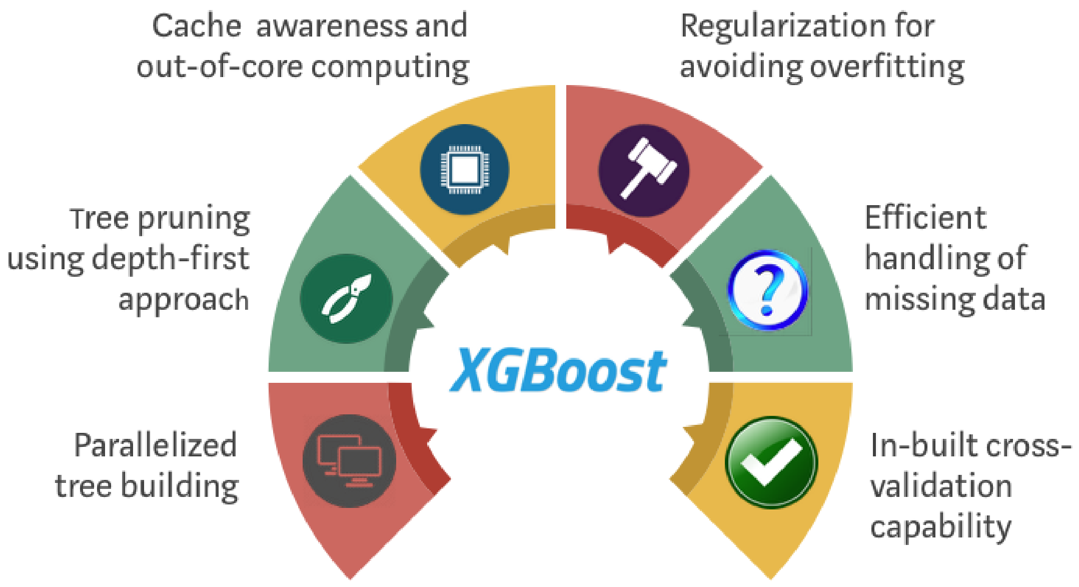
- is short for Extreme Gradient Boosting - the most popular **implementation** of gradient boosting.

  💡 XGBoost is the winning solution for many Kaggle competitions.

- According to **XGBoost Documentation**

  > XGBoost is an optimized distributed gradient boosting library designed to be highly **efficient**, **flexible** and **portable**. The goal of this library is to push the extreme of the computation limits of machines to provide a **scalable**, **portable** and **accurate** library.
  >
  > XGBoost provides a parallel tree boosting that solve many data science problems in a fast and accurate way. The same code runs on major distributed environment (Hadoop, SGE, MPI) and can solve problems beyond **billions** of examples.

- It integrates several approximations and tricks that speed up the training process significantly.
    - **Algorithm enhancements**
        - Minimizes a regularized (L1 and L2) objective function that combines a convex loss function and a penalty term for model complexity → avoid overfitting
        - Efficient handling of missing data → simply data preprocessing
        - Built-in cross-validation capability (at each iteration) → prevents the need to calculate the number of boosting iterations needed
    - **System optimization** → increase speed
        - Parallelized tree building → increase speed
        - Tree pruning using ‘depth-first’ approach and it prunes the tree in a backward direction (unlike the stopping criterion for tree splitting used by GBMs)
        - Hardware optimization (out-of-core computing)


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

Credit: Saksham Gulati, https://sakshamgulati123.medium.com/xgboost-4cb311310adb