Michio Suginoo
Email: msuginoo@deeporigami.com
LinkedIn: https://www.linkedin.com/in/reversalpoint/
Github: https://github.com/deeporigami/

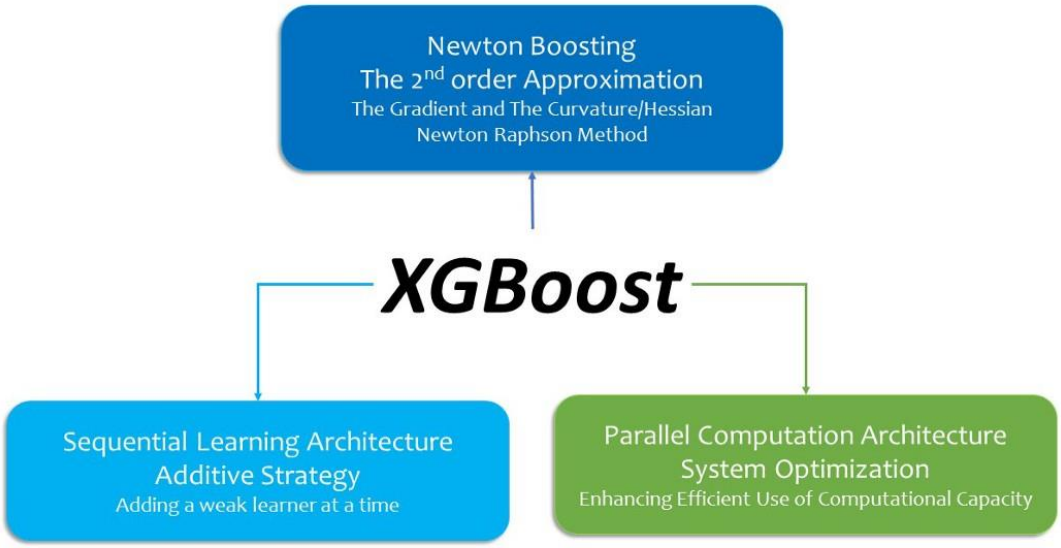
XGBoost

Its Genealogy, its Architectural Features, and its Innovations

Sequential and Parallel Architectures All in One

Michio Suginoo

3 October, 2022



XGBoost (eXtreme **G**radient **Boost**) is a powerful learning algorithm which had outperformed many conventional Machine Learning algos in many competitions in the past.

In a nutshell, ***XGBoost*** is equipped with both sequential and parallel architectures all in one: while it is a sequential learning algorithm (additive strategy), it incorporates parallel computation into its architecture in order to enhance the system efficiency.

This reading is Part 1 of my project note of Hyperparameter Tuning of XGBoost and covers a quick overview of the learning algorithm. Part 2 will go over the implementation of the hyperparameter tuning of ***the native XGBoost API*** in the context of a regression use case with California Housing Price dataset.

Michio Suginoo
Email: msuginoo@deeporigami.com
LinkedIn: https://www.linkedin.com/in/reversalpoint/
Github: https://github.com/deeporigami/

Those familiar with the architecture of XGBoost can skip this reading and move on to Part2. For those who are not familiar with the learning algorithm, this post will serve a one-stop reading material that would give you a big picture about XGBoost—its genealogy, its architectural features, and its innovative features. At the end I will also suggest a shorthand list of supplemental readings so that the readers can explore more details of the topics covered in the reading.

Now, let's start.

In order to understand the features of XGBoost, we can start with a quick overview of its genealogy.

A. Genealogy of XGBoost

From a top-down perspective, XGBoost is a sub-class of ***Supervised Machine Learning***. And, as its name suggests, XGBoost is an advanced variant of ***Boosting Machine***, which is a sub-class of ***Tree-based Ensemble*** algorithm, like Random Forest.

Nevertheless, Boosting Machine is fundamentally different from Random Forest in the way how it operates its learning processes.

Boosting Machine

Random Forest runs multiple independent decision trees in parallel and combines their results by averaging all the results. This approach uses random bootstrapping sampling and is often called 'bagging'. In this sense, Random Forest is a parallel learning algorithm.

On the contrary, Boosting Machine uses "*an additive strategy*": that is to "*add one new tree at a time*". Boosting Machine runs individual weak/simple decision trees called ***"the base learner"*** in sequence. In this sense, Boosting Machine learns in sequence, while Random Forest does in parallel. Simply put, conceptually Boosting Machine is built on a sequential learning architecture.

For a further reference on Boosting Machine, here is a MIT lecture on Boosting:
https://www.youtube.com/watch?v=UHBmv7qCey4

That said, to avoid confusion I should make a footnote here from the perspective of ***system optimization***. XGBoost is also designed to operate parallel computation to enhance an efficient use of computational resources[1]. Overall, XGBoost, while inheriting a sequential learning architecture from Boosting Machine, operates parallel computations for System Optimization.

Gradient Boosting Machine

Michio Suginoo
Email: msuginoo@deeporigami.com
LinkedIn: https://www.linkedin.com/in/reversalpoint/
Github: https://github.com/deeporigami/

As its name suggests, XGBoost (eXtreme Gradient Boost) is an advanced variant of Gradient Boosting Machine (GBM), a family member of Boosting Machine.

As a part of its ***additive strategy***, ***Gradient Boosting Machine (GBM***) uses Gradient Descent for optimization. In other words, it uses ***the first derivative*** (***Gradient***) of the Objective Function (Loss Function) to determine the next weak learner predictor. In this way, Gradient Boosting, while retaining the existing weak predictors, adds a new predictor on top of them to reduce the current error in order to incrementally improve the performance. Moreover, in order to reduce the computational burden, GBM approximates the Objective Function by using the first order term of the Taylor expansion and ignores any higher order terms for its learning optimization. (Friedman, 2000)

B. Algorithmic Features of XGBoost

Newton Boosting Machine

XGBoost extends the idea of Gradient Boosting in the sense that it also uses the second derivative (***Hessian: Curvature***) of the Objective Function in addition to its first derivative (Gradient) to further optimize its learning process. The method is called ***the Newton Raphson Method***. And Boosting Machine using the Newton Raphson Method is called ***Newton Boosting***. For further discussions on the difference between the Gradient Descent and the Newton Boosting, here is a good reading: https://arxiv.org/abs/1808.03064.

Because of the specific architecture of the additive strategy, the second order approximation yields multiple beneficial mathematical properties to streamline the algorithm for further computational efficiency. (Guestrin & Chen, 2016)

Regularization: to address "Variance-Bias Trade-off"

Jerome Friedman, the architect of Gradient Boosting Machine (Friedman, 2000), articulated the importance of regularization to address ***"variance-bias trade-off"***, the problem of underfitting-overfitting trade-off[2], specifically recommending the users to tune three meta-parameters of Gradient Boosting Machine: the number of iterations[3], the learning rate, and the number of terminal nodes/leaves[4].

In short, XGBoost inherited the regularization focus of Gradient Boosting Machine and extended it further.

- First, XGBoost enables the users to tune the various hyperparameters to constrain the trees: e.g. the number of trees, the depth of an individual tree, the minimum sum of instance weights for partition, the maximum number of boosting rounds, and the number of the nodes/leaves.

- Second, it allows the users to apply shrinkage, "learning rate", during the learning process[5].
- Third, it enables the users to use random sampling techniques such as "column sub-sampling"[6].
- Fourth, it enables the users to tune L1 and L2 regularization terms.

C. Innovations:

Sparsity-aware Algorithm and Weighted Quantile Sketch

More importantly, XGBoost introduced two innovations[7]. This should not be understated.

One is a sparsity aware algorithm that has a feature called "default direction" to determine the direction of the split at each node to handle sparse data structure. In principle, thanks to this feature, XGBoost can handle missing data: the user does not need to impute missing data. Here is the explanation about data sparsity and "default direction" by Chen and Guestrin.

> *"There are multiple possible causes for sparsity: 1) presence of missing values in the data; 2) frequent zero entries in the statistics; and, 3) artifacts of feature engineering such as one-hot encoding. It is important to make the algorithm aware of the sparsity pattern in the data. In order to do so, we propose to add a default direction in each tree node, which is shown in Fig. 4. When a value is missing in the sparse matrix x, the instance is classified into the default direction. There are two choices of default direction in each branch. The optimal default directions are learnt from the data." (Guestrin & Chen, 2016)*

The other innovation is a ***weighted quantile sketch***. The following excerpt from Chen and Guestrin's paper summarizes what it is.

> *"One important step in the approximate algorithm is to propose candidate split points. Usually percentiles of a feature are used to make candidates distribute evenly on the data. ... However, there is no existing quantile sketch for the weighted datasets. Therefore, most existing approximate algorithms either resorted to sorting on a random subset of data which have a chance of failure or heuristics that do not have theoretical guarantee. To solve this problem, we introduced a novel distributed weighted quantile sketch algorithm that can handle weighted data with a provable theoretical guarantee. The general idea is to propose a data structure that supports merge and prune operations, with each operation proven to maintain a certain accuracy level." (Guestrin & Chen, 2016)*

System Optimization: Efficiency and Scalability

Michio Suginoo
Email: msuginoo@deeporigami.com
LinkedIn: https://www.linkedin.com/in/reversalpoint/
Github: https://github.com/deeporigami/

So far, we saw the framework of XGBoost from the learning algorithm's architectural perspective. Now, we can view it from the perspective of System Optimization.

The native XGBoost API is also innovative in pursuing the system optimization. The API is called "eXtreme" (X) since XGBoost aims at enabling the users to exploit an eXtreme limit of the given system's computational capacity, by efficiently allocating computation tasks among the given computational resources[8]—processors (CPU, GPU), memory, and out-of-core (disk space): cache access, block data compression and sharding.

On more about the innovative aspects of the native XGBoost API, here is a great piece outlined by the inventors (Chen & Guestrin) , "*XGBoost: A Scalable Tree Boosting System*": https://arxiv.org/abs/1603.02754.

D. Ending Remark

This quick overview of XGBoost went over its genealogy, its architectural features, and its innovation without getting into details.

Overall, XGBoost is equipped with both sequential (additive strategy) architecture for learning and parallel computation for the system optimization.

Along its genealogy, it is a Supervised Machine Learning algorithm, a family member of Tree Ensemble, and an advanced variant of Gradient Boosting Machine (GBM).

It extended GBM by using not only the first derivative (***Gradient***) but also the second derivative (***Curvature; Hessian***) for the approximation (***Newton-Raphson Method***) of the objective function. In this sense, it is a ***Newton Boosting Machine***.

It inherited the spirit of GBM in making an intense focus on regularization to address "***bias-variance trade-off***", the problem of underfitting-overfitting trade-off. It has a built-in feature called "default direction" to handle sparse data structure, such as missing data.

Below, I would like to mention few good resources about XGBoost.

- For a quick overview of XGBoost, "A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning" by Jason Brownlee is very concise in capturing the genealogy and major features of XGBoost: https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/
- For mathematical explanation of XGBoost, "TreeBoosting-03: Why Does XGBoost Win Every Machine Learning Competition?" by Ha NGUYEN gave

me a good supplemental resource: https://datasciblog.github.io/2020/02/26/tree-boosting-03/

- For the innovative side of XGBoost, "*XGBoost: A Scalable Tree Boosting System*" by the inventor of XGBoost, Chen & Guestrin, will give you a brief summary: https://arxiv.org/abs/1603.02754
- The official native XGBoost API's online documentation gives you an official tutorial. "XGBoost Tutorials" is a great base resource of the innovative algorithm: https://xgboost.readthedocs.io/en/latest/tutorials/index.html
- If you have time and soul to read a heavy load paper, you should read Jerome H. Friedman's paper on Gradient Boosting Machine, "Greedy Function Approximation: A Gradient Boosting Machine": https://jerryfriedman.su.domains/ftp/trebst.pdf

by Michio Suginoo

Footnote

[1] "*XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) 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.*" (Source: https://xgboost.readthedocs.io/en/stable/)

[2] "*In prediction problems, fitting the training data too closely can be counterproductive. (…) Reducing the expected loss on the training data beyond some point causes the population-expected loss to stop decreasing and often to start increasing. Regularization methods attempt to prevent such "overfitting" by constraining the fitting procedure.*" (Friedman, 2000, p. 1203)

[3] "*For additive expansions a natural regularization parameter is the number of components M. … Controlling the value of M regulates the degree to which expected loss on the training data can be minimized. The best value for M can be estimated by some model selection method, such as using an independent "test" set, or cross-validation. Regularizing by controlling the number of terms in the expansion places an implicit prior belief that "sparse" approximations involving fewer terms are likely to provide better prediction."* (Friedman, 2000, p. 1203)

[4] "*there are the metaparameters associated with the procedure used to estimate the base learner (…) The primary focus of this paper has been on the use of best-first induced regression trees with a fixed number of terminal nodes, J. Thus, J is the primary metaparameter of this base learner. The best choice for its value depends most strongly on the nature of the target function, namely the highest order of the dominant*

interactions among the variables. (…) The highest interaction order possible is limited by the number of input variables ***n****. (…) In boosting regression trees, the interaction order can be controlled by limiting the size of the individual trees induced at each iteration. A tree with* ***J*** *terminal nodes produces a function with interaction order at most min(****J****—1,* ***n****). The boosting process is additive, so the interaction order of the entire approximation can be no larger than the largest among its individual components. Therefore, with any of the TreeBoost procedures, the best tree size* ***J*** *is governed by the effective interaction order of the target (function). This is usually unknown so that J becomes a metaparameter of the procedure to be estimated using a model selection criterion such as cross-validation or on a left-out subsample not used in training. However, as discussed above, it is unlikely that large trees would ever be necessary or desirable."* (Friedman, 2000, pp. 1214–1215)

[5] "*shrinkage reduces the influence of each individual tree and leaves space for future trees to improve the model*." (Guestrin & Chen, 2016, p. 3)

[6] "*According to user feedback, using column sub-sampling prevents over-fitting even more so than the traditional row sub-sampling (which is also supported). The usage of column sub-samples also speeds up computations of the parallel algorithm described later*." (Guestrin & Chen, 2016, p. 3)

[7] "*We proposed a novel sparsity aware algorithm for handling sparse data and a theoretically justified weighted quantile sketch for approximate learning. Our experience shows that cache access patterns, data compression and sharding are essential elements for building a scalable end-to-end system for tree boosting.*" (Chen & Guestrin, 2016)

[8] *"More importantly, it is developed with both deep consideration in terms of* ***systems optimization*** *and* ***principles in machine learning****. The goal of this library is to push the extreme of the computation limits of machines to provide a* ***scalable****,* ***portable*** *and* ***accurate*** *library."* (databricks, 2017)

References

- Brownlee, J. *A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning*. (2016). Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/
- databricks. *xgboost-linux64*. (2017). Retrieved from github: https://github.com/databricks/xgboost-linux64/blob/master/doc/model.md
- Friedman, J. *Greedy Function Approximation: A Gradient Boosting Machine*. (2000). *the Annals of Statistics, 29*(5), 1189–1232. doi:10.1214/aos/1013203451
- Guestrin, C., & Chen, T. *XGBoost: A Scalable Tree Boosting System*. (2016). doi: https://doi.org/10.48550/arXiv.1603.02754

- Nguyen, H. *TreeBoosting-03: Why Does XGBoost Win Every Machine Learning Competition?* (n.d.). Retrieved from Data Science Blog: https://datasciblog.github.io/2020/02/26/tree-boosting-03
- xgboost developers. *XGBoost Documentation*. (n.d.). Retrieved from https://xgboost.readthedocs.io/: https://xgboost.readthedocs.io/en/stable/