XGBoost Algorithm: Long May She Reign!

The new queen of Machine Learning algorithms taking over the world…

Vishal Morde · Follow
Published in Towards Data Science
7 min read · Apr 8, 2019

Listen Share

(This article was co-authored with Venkat Anurag Setty)

I still remember the day 1 of my very first job fifteen years ago. I had just finished my graduate studies and joined a global investment bank as an analyst. On my first day, I kept straightening my tie and trying to remember everything that I had studied. Meanwhile, deep down, I wondered if I was good enough for the corporate world. Sensing my anxiety, my boss smiled and said:

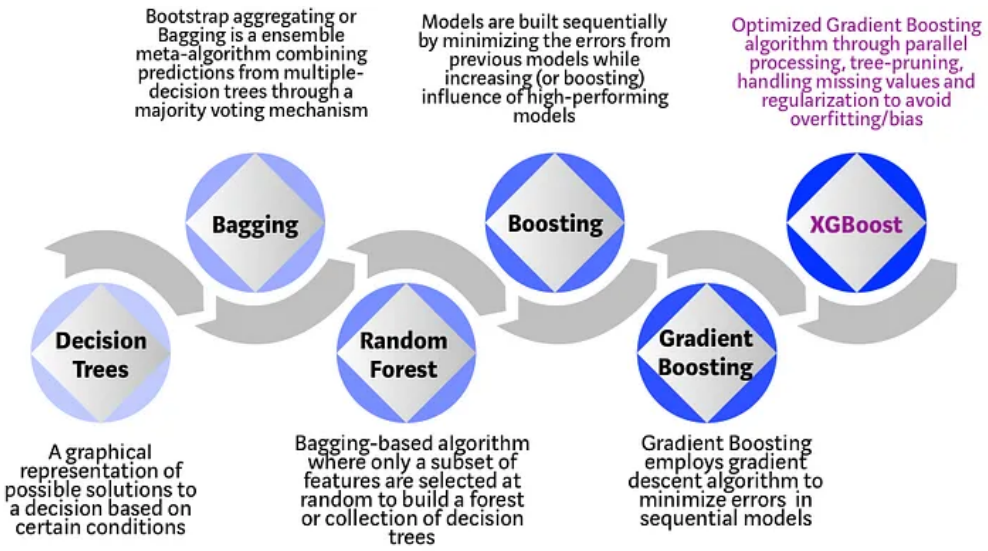
"Don't worry! The only thing that you need to know is the regression modeling!"

I remember thinking myself, "I got this!". I knew regression modeling; both linear and logistic regression. My boss was right. In my tenure, I exclusively built regression-based statistical models. I wasn't alone. In fact, at that time, regression modeling was the undisputed queen of predictive analytics. Fast forward fifteen years, the era of regression modeling is over. The old queen has passed. Long live the new queen with a funky name; XGBoost or Extreme Gradient Boosting!

. . .

What is XGBoost?

XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. In prediction problems involving unstructured data (images, text, etc.) artificial neural networks tend to outperform all other algorithms or frameworks. However, when it comes to small-to-medium structured/tabular data, decision tree based algorithms are considered best-in-class right now. Please see the chart below for the evolution of tree-based algorithms over the years.



Evolution of XGBoost Algorithm from Decision Trees

XGBoost algorithm was developed as a research project at the University of Washington. Tianqi Chen and Carlos Guestrin presented their paper at SIGKDD Conference in 2016 and caught the Machine Learning world by fire. Since its introduction, this algorithm has not only been credited with winning numerous Kaggle competitions but also for being the driving force under the hood for several cutting-edge industry applications. As a result, there is a strong community of data scientists contributing to the XGBoost open source projects with ~350 contributors and ~3,600 commits on GitHub. The algorithm differentiates itself in the following ways:

1. A wide range of applications: Can be used to solve regression, classification, ranking, and user-defined prediction problems.
2. Portability: Runs smoothly on Windows, Linux, and OS X.
3. Languages: Supports all major programming languages including C++, Python, R, Java, Scala, and Julia.
4. Cloud Integration: Supports AWS, Azure, and Yarn clusters and works well with Flink, Spark, and other ecosystems.

. . .

How to build an intuition for XGBoost?

Decision trees, in their simplest form, are easy-to-visualize and fairly interpretable algorithms but building intuition for the next-generation of tree-based algorithms can be a bit tricky. See below for a simple analogy to better understand the evolution of tree-based algorithms.

Photo by rawpixel on Unsplash

Imagine that you are a hiring manager interviewing several candidates with excellent qualifications. Each step of the evolution of tree-based algorithms can be viewed as a version of the interview process.

1. **Decision Tree**: Every hiring manager has a set of criteria such as education level, number of years of experience, interview performance. A decision tree is analogous to a hiring manager interviewing candidates based on his or her own criteria.

2. **Bagging**: Now imagine instead of a single interviewer, now there is an interview panel where each interviewer has a vote. Bagging or bootstrap aggregating involves combining inputs from all interviewers for the final decision through a democratic voting process.

3. **Random Forest**: It is a bagging-based algorithm with a key difference wherein only a subset of features is selected at random. In other words, every interviewer will only test the interviewee on certain randomly selected qualifications (e.g. a technical interview for testing programming skills and a behavioral interview for evaluating non-technical skills).

4. **Boosting**: This is an alternative approach where each interviewer alters the evaluation criteria based on feedback from the previous interviewer. This 'boosts' the efficiency of the interview process by deploying a more dynamic evaluation process.

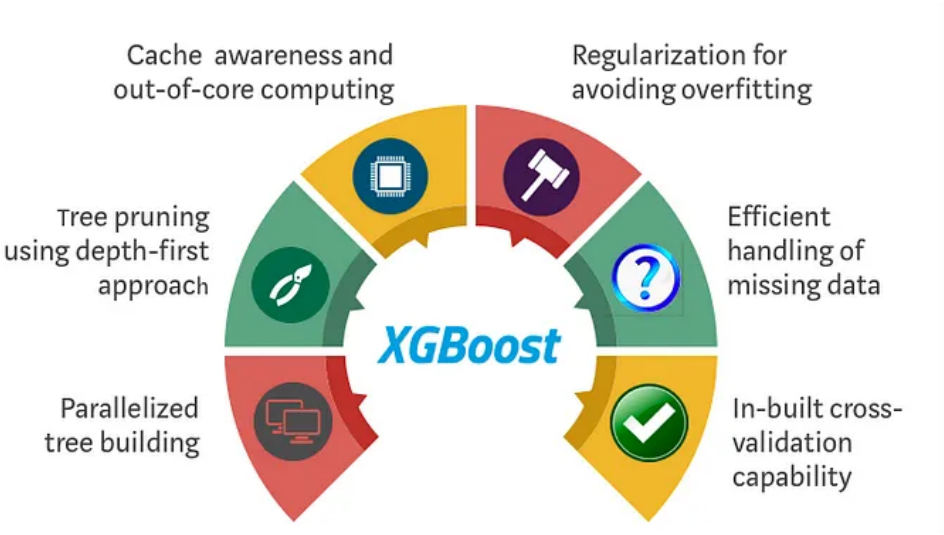
5. **Gradient Boosting**: A special case of boosting where errors are minimized by gradient descent algorithm e.g. the strategy consulting firms leverage by using case interviews to weed out less qualified candidates.

6. **XGBoost**: Think of XGBoost as gradient boosting on 'steroids' (well it is called 'Extreme Gradient Boosting' for a reason!). It is a perfect combination of software and hardware optimization techniques to yield superior results using less computing resources in the shortest amount of time.

. . .

Why does XGBoost perform so well?

XGBoost and Gradient Boosting Machines (GBMs) are both ensemble tree methods that apply the principle of boosting weak learners (CARTs generally) using the gradient descent architecture. However, XGBoost improves upon the base GBM framework through systems optimization and algorithmic enhancements.



How XGBoost optimizes standard GBM algorithm

System Optimization:

1. **Parallelization**: XGBoost approaches the process of sequential tree building using parallelized implementation. This is possible due to the interchangeable nature of loops used for building base learners; the outer loop that enumerates the leaf nodes of a tree, and the second inner loop that calculates the features. This nesting of loops limits parallelization because without completing the inner loop (more computationally demanding of the two), the outer loop cannot be started. Therefore, to improve run time, the order of loops is interchanged using initialization through a global scan of all instances and sorting using parallel threads. This switch improves algorithmic performance by offsetting any parallelization overheads in computation.

2. **Tree Pruning:** The stopping criterion for tree splitting within GBM framework is greedy in nature and depends on the negative loss criterion at the point of split. XGBoost uses 'max_depth' parameter as specified instead of criterion first, and starts pruning trees backward. This 'depth-first' approach improves computational performance significantly.

3. **Hardware Optimization**: This algorithm has been designed to make efficient use of hardware resources. This is accomplished by cache awareness by allocating internal buffers in each thread to store gradient statistics. Further enhancements such as 'out-of-core' computing optimize available disk space while handling big data-frames that do not fit into memory.

Algorithmic Enhancements:

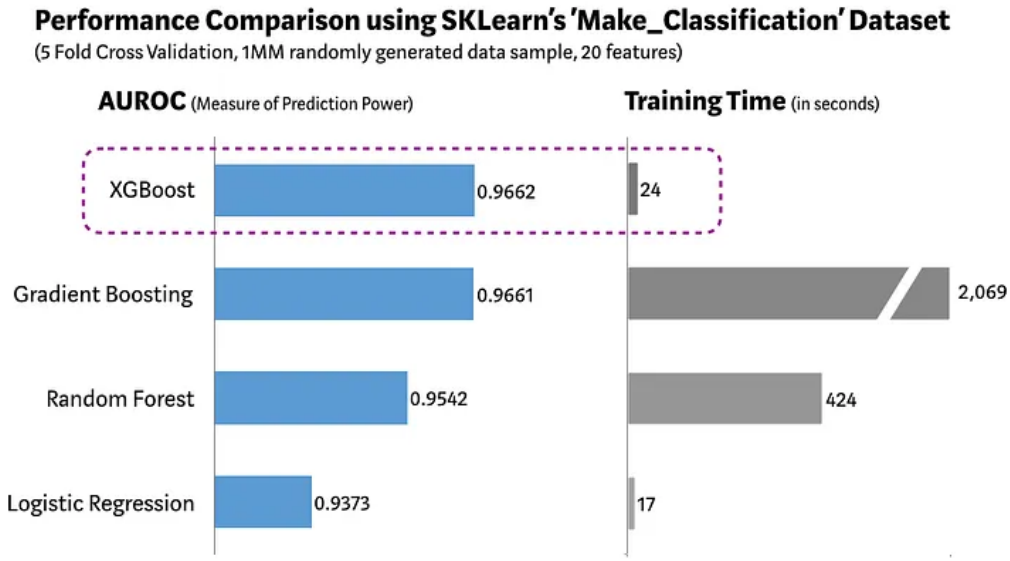
1. **Regularization**: It penalizes more complex models through both LASSO (L1) and Ridge (L2) regularization to prevent overfitting.

2. **Sparsity Awareness**: XGBoost naturally admits sparse features for inputs by automatically 'learning' best missing value depending on training loss and handles different types of sparsity patterns in the data more efficiently.
3. **Weighted Quantile Sketch:** XGBoost employs the distributed weighted Quantile Sketch algorithm to effectively find the optimal split points among weighted datasets.
4. **Cross-validation**: The algorithm comes with built-in cross-validation method at each iteration, taking away the need to explicitly program this search and to specify the exact number of boosting iterations required in a single run.

. . .

Where is the proof?

We used Scikit-learn's 'Make_Classification' data package to create a random sample of 1 million data points with 20 features (2 informative and 2 redundant). We tested several algorithms such as Logistic Regression, Random Forest, standard Gradient Boosting, and XGBoost.



XGBoost vs. Other ML Algorithms using SKLearn's Make_Classification Dataset

As demonstrated in the chart above, XGBoost model has the best combination of prediction performance and processing time compared to other algorithms. Other rigorous benchmarking studies have produced similar results. No wonder XGBoost is widely used in recent Data Science competitions.

> *"When in doubt, use XGBoost" — Owen Zhang, Winner of Avito Context Ad Click Prediction competition on Kaggle*

. . .

So should we use just XGBoost all the time?

When it comes to Machine Learning (or even life for that matter), there is no free lunch. As Data Scientists, we must test all possible algorithms for data at hand to identify the champion algorithm. Besides, picking the right algorithm is not enough. We must also choose the right configuration of the algorithm for a dataset by tuning the hyper-

parameters. Furthermore, there are several other considerations for choosing the winning algorithm such as computational complexity, explainability, and ease of implementation. This is exactly the point where Machine Learning starts drifting away from science towards art, but honestly, that's where the magic happens!

. . .

What does the future hold?

Machine Learning is a very active research area and already there are several viable alternatives to XGBoost. Microsoft Research recently released LightGBM framework for gradient boosting that shows great potential. CatBoost developed by Yandex Technology has been delivering impressive bench-marking results. It is a matter of time when we have a better model framework that beats XGBoost in terms of prediction performance, flexibility, explanability, and pragmatism. However, until a time when a strong challenger comes along, XGBoost will continue to reign over the Machine Learning world!