STEVENS INSTITUTE OF TECHNOLOGY

XGBoost

Rensheng Wang,
https://sit.instructure.com/courses/55957

March 3, 2022

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.

What is XGBoost?

- XGBoost is short for eXtreme Gradient Boosting.
- It is an open-sourced tool:
  - ComputationinC++
  - R/Python/Julia interface provided
- A wide range of applications: Can be used to solve regression, classification, ranking, and user-defined prediction problems.
- A variant of the gradient boosting machine
  - Tree-based model
- Cloud Integration: Supports AWS, Azure, and Yarn clusters and works well with Flink, Spark, and other ecosystems.
- The winning model for several kaggle competitions

Intuition for XGBoost?

- Decision trees, in their simplest form, are easy-to-visualize and fairly interpretable algorithms
- However, building intuition for the next generation of tree-based algorithms can be a bit tricky.
- ☞ 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.

Intuition for XGBoost?

- 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.

- 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.

- Random Forest:

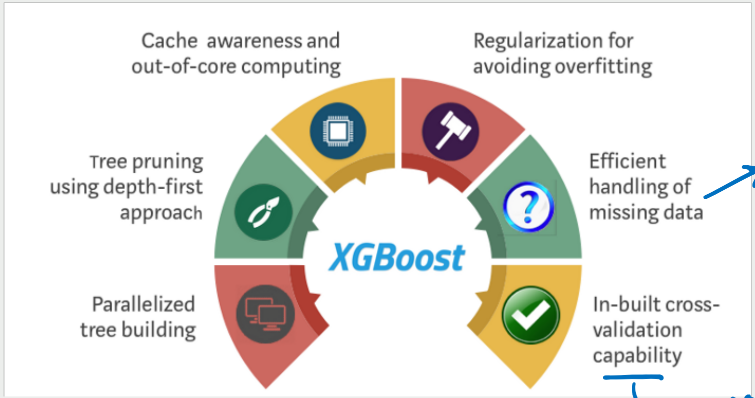
  It is a bagging-based algorithm with a key difference wherein only a subset of features is selected at random. In other words, every interviewer will only test the interviewee on certain randomly selected qualifications (e.g. a technical interview for testing programming skills and a behavioral interview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

Intuition for XGBoost?

- 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.
- 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.
- 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 XGBoost performs 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.


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

→ manual cleaning

↳ no worry about overfitting

Why XGBoost performs so well

- 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.

Why XGBoost performs so well

- System Optimization:

  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.

Why XGBoost performs so well

- 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:

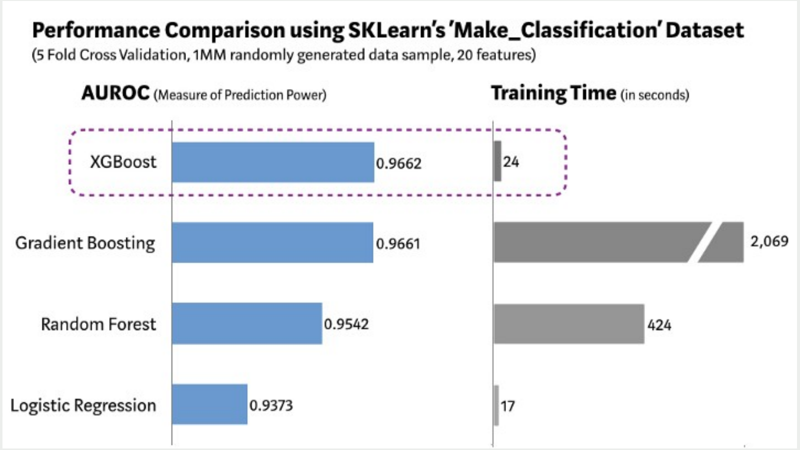
     XGBoost employs the distributed weighted Quantile Sketch algorithm to effectively find the optimal split points among weighted datasets.

  4. Cross-validation:

     The algorithm comes with built-in cross-validationmethod 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.



So Use XGBoost All theTime?

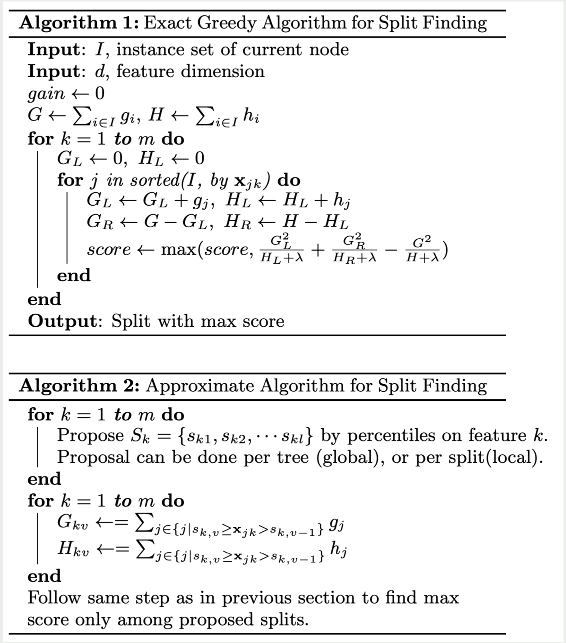
- When it comes to Machine Learning (or even life for that matter), there is no free lunch.
- As Data Scientists, we must test all possible algorithms for data at hand to identify the champion algorithm.
- Besides, picking the right algorithm is not enough. We must also choose the right configuration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

Introduction of XGBoost

XGBoost is widely used for Kaggle competitions. The reason to choose XGBoost includes

- Easy to use
  - Easy to install
  - Highly developed R/Python interface for users
- Efficiency
  - Automatic parallel computation on a single machine
  - Can be run on a cluster
- Accuracy
  - Good result for most data sets
- Feasibility
  - Customized objective and evaluation
  - Tunable parameters

Algorithms



```
Algorithm 1: Exact Greedy Algorithm for Split Finding
  Input: $I$, instance set of current node
  Input: $d$, feature dimension
  $gain \leftarrow 0$
  $G \leftarrow \sum_{i \in I} g_i$, $H \leftarrow \sum_{i \in I} h_i$
  for $k = 1$ to $m$ do
      $G_L \leftarrow 0,\ H_L \leftarrow 0$
      for $j$ in sorted($I$, by $\mathbf{x}_{jk}$) do
          $G_L \leftarrow G_L + g_j,\ H_L \leftarrow H_L + h_j$
          $G_R \leftarrow G - G_L,\ H_R \leftarrow H - H_L$
          $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
      end
  end
  Output: Split with max score
```

```
Algorithm 2: Approximate Algorithm for Split Finding
  for $k = 1$ to $m$ do
      Propose $S_k = \{s_{k1}, s_{k2}, \cdots s_{kl}\}$ by percentiles on feature $k$.
      Proposal can be done per tree (global), or per split(local).
  end
  for $k = 1$ to $m$ do
      $G_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} g_j$
      $H_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} h_j$
  end
  Follow same step as in previous section to find max
  score only among proposed splits.
```

Regularized Learning Objective

- For a given data set with n examples and m features $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$ $(|\mathcal{D}| = n, \mathbf{x}_i \in \mathbb{R}^m, y_i \in \mathbb{R})$, a tree ensemble model uses K additive functions to predict the output

$$\hat{y}_i = \phi(\mathbf{x}_i) = \sum_{k=1}^{K} f_k(\mathbf{x}_i), \qquad f_k \in \mathcal{F}$$

where $\mathcal{F} = \{f(\mathbf{x} = w_q(\mathbf{x})\}(q : \mathbb{R}^m \to T, w \in \mathbb{R}^T)$ is the space of regression trees (also known as CART(Classification and Regression Trees)).

  - ☞ q : the structure of each tree that maps an example to the corresponding leaf index
  - ☞ T : the number of leaves in the tree.
  - ☞ f_k : an independent tree structure q and leaf weights w.

- Unlike decision trees, each regression tree contains a continuous score on each of the leaf, we use w_i to represent score on i-th leaf

Regularized Learning Objective

- Unlike decision trees, each regression tree contains a continuous score on each of the leaf, we use w_i to represent score on i-th leaf
- For a given example, we will use the decision rules in the trees (given by q) to classify it into the leaves and calculate the final prediction by sum- ming up the score in the corresponding leaves (given by w)
- To learn the set of functions used in the model, we minimize the following regularized objective:

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i,\ y_i) + \sum_k \Omega(f_k)$$

where $\Omega(f) = \gamma T + \frac{1}{2}\|w\|^2$

- Here l is a differentiable convex loss function that measures the difference between the prediction $\hat{y}_i$ and the target y_i. The second term Ω penalizes the complexity of the model (i.e., the regression tree functions).

Gradient Tree Boosting

- The objective function to minimize:

$$\mathcal{L}^{(t)} \sum_{i=1}^{n} l\left(y_i,\ \hat{y}_i^{(t-1)} + f_t(\mathbf{x}_i)\right) + \Omega(f_t)$$

- Second-order approximation can be used to quickly optimize the objective

$$\mathcal{L}^{(t)} \approxeq \sum_{i=1}^{n} \left[l(y_i,\ \hat{y}^{(t-1)}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i) \right] + \Omega(f_t)$$

where $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$ and $h_i = \partial^2_{\hat{y}^{(-1)}} l(y_i, \hat{y}^{(t-1)})$ are the first and second order gradient statistics on the loss function.

Gradient Tree Boosting

- Remove the constant term:

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} \left[g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i) \right] + \Omega(f_t)$$

- Define $I_j = \{i | q(\mathbf{x}_i = j\}$ as the instance set of leaf j and rewrite the equation as

$$\begin{aligned} \tilde{\mathcal{L}}^{(t)} &= \sum_{i=1}^{n} \left[g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i) \right] + \Omega(f_t) \\ &= \sum_{i=1}^{n} \left[g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i) \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T} \left[(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2 \right] + \gamma T \end{aligned}$$

Sparsity-aware Split Finding

- If the input $\mathbf{X}$ to be sparse, it might be: 1.) Presence of missing values; 2.) frequent zero entries in the statistics; 3.) artifacts of feature engineering such as one-hot encoding.
- To make the algorithm aware of the sparsity pattern in the data, add a default direction in each tree node
- When a value is missing in the sparse matrix $\mathbf{X}$, the instance is classified into the default direction.
- There are two choices of default direction in each branch. The optimal default di- rections are learnt from the data.

from XGBoost to LightGBM

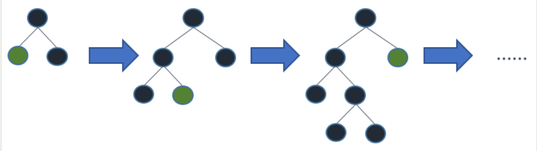
- A couple of years ago, Microsoft announced its gradient boosting framework LightGBM. Nowadays, it steals the spotlight in gradient boosting machines. Kagglers start to use LightGBM more than XGBoost. LightGBM is 6 times faster than XGBoost.
- LightGBM is a gradient boosting framework that uses tree-based learning algorithms. It is designed to be distributed and efficient with the following advantages:
  - Faster training speed and higher efficiency.
  - Lower memory usage.
  - Better accuracy.
  - Support of parallel and GPU learning.
  - Capable of handling large-scale data.

LightGBM intuition

- LightGBM is a gradient boosting framework that uses tree-based learning algorithm.
- LightGBM grows tree vertically while other tree-based learning algorithms grow trees horizontally.
- It means that LightGBM grows tree leaf-wise while other algorithms grow level-wise.
- It will choose the leaf with max delta loss to grow. When growing the same leaf, leaf-wise algorithm can reduce more loss than a level-wise algorithm.

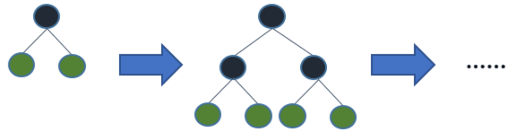
Leaf-Wise vs.Level-Wise Tree Growth

- Leaf-wise tree growth can best be explained with the following visual



Leaf-wise tree growth

- Most decision tree learning algorithms grow tree by level (depth)-wise
  Level-wise tree growth can best be explained with the following visual



Level-wise tree growth

Important Points about Tree-growth

- If we grow the full tree, best-first (leaf-wise) and depth-first (level-wise) will result in the same tree. The difference is in the order in which the tree is expanded. Since we donot normally grow trees to their full depth, order matters.
- Application of early stopping criteria and pruning methods can result in very different trees. Because leaf-wise chooses splits based on their contribution to the global loss and not just the loss along a particular branch, it often (not always) will learn lower-error trees “faster” than level-wise.
- For a small number of nodes, leaf-wise will probably out-perform level-wise. As we add more nodes, without stopping or pruning they will converge to the same performance because they will literally build the same tree eventually.

XGBoost Vs LightGBM

- XGBoost is a very fast and accurate ML algorithm. But now it's been challenged by LightGBM which runs even faster with comparable model accuracy and more hyperparameters for users to tune.

- The key difference in speed is because XGBoost split the tree nodes one level at a time and LightGBM does that one node at a time.

  So XGBoost developers later improved their algorithms to catch up with LightGBM, allowing users to also run XGBoost in split-by-leaf mode (grow_policy = 'lossguide'). Now XGBoost is much faster with this improvement, but LightGBM is still about 1.3X $\sim$ 1.5X the speed of XGB.

- Another difference between XGBoost and LightGBM is that XGBoost has a feature that LightGBM lacks monotonic constraint. It will sacrifice some model accuracy and increase training time, but may improve model interpretability.

Novel Techniques

- Gradient-based One-Side Sampling (GOSS)

  GOSS excludes a significant proportion of data instances with small gradients, and only use the rest to estimate the information gain.

  It has been proved that, since the data instances with larger gradients play a more important role in the computation of information gain, GOSS can obtain quite accurate estimation of the information gain with a much smaller data size.

- Exclusive Feature Bundling (EFB)

  Bundle mutually exclusive features (i.e., they rarely take nonzero values simultaneously), to reduce the number of features.

  It has been proved that finding the optimal bundling of exclusive features is NP-hard, but a greedy algorithm can achieve quite good approximation ratio (and thus can effectively reduce the number of features without hurting the accuracy of split point determination by much).

XGBoost Python

Here is the Python code:

```
# Regression
from sklearn import datasets
X,y = datasets.load_diabetes(return_X_y=True)
from xgboost import XGBRegressor
from sklearn.model_selection import cross_val_score
scores = cross_val_score(XGBRegressor(objective='reg:squarederror'), \
    X, y, scoring='neg_mean_squared_error')
(-scores)**0.5

# Classifier
# you can define your own data X, y
from xgboost import XGBClassifier
from sklearn.model_selection import cross_val_score
cross_val_score(XGBClassifier(), X, y)
```

LightGBM Python

Here is the Python code:

```python
# split the dataset into the training set and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, \
    random_state = 0)

# build the lightgbm model
import lightgbm as lgb
clf = lgb.LGBMClassifier()
clf.fit(X_train, y_train)
```