Decision Trees and XGBOOST Algorithms

Table of Contents

Decision Trees and XGBOOST Algorithms

Introduction

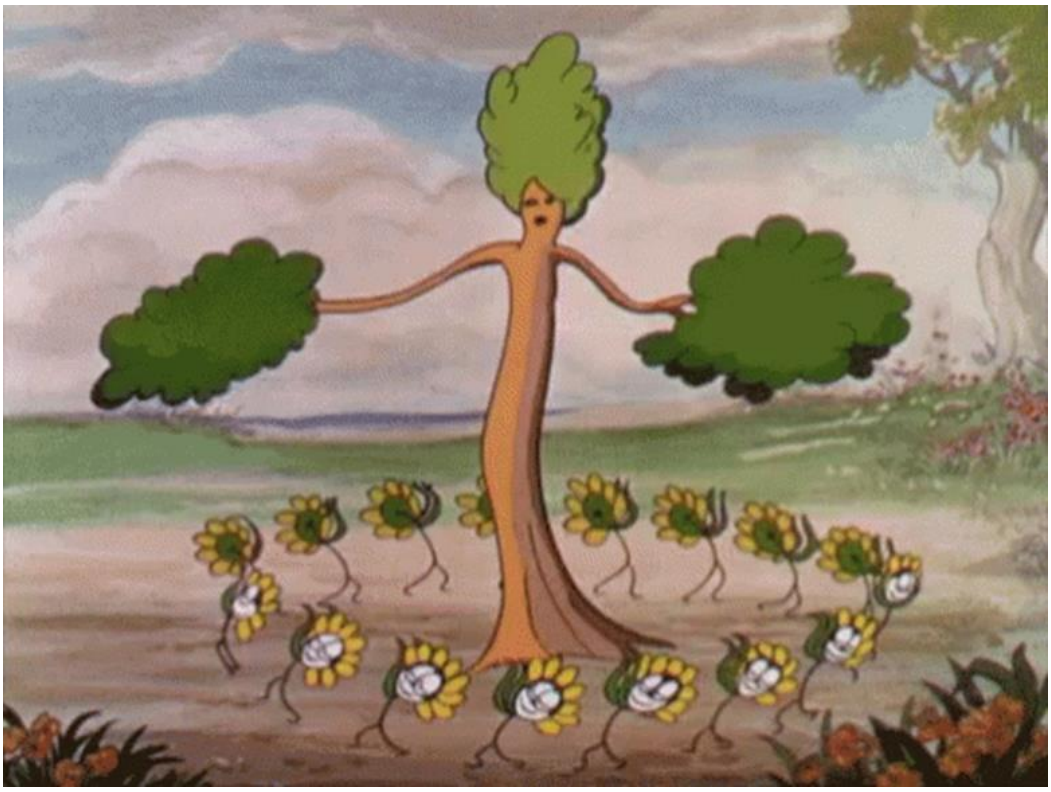


Fig 1: Decision Trees

(Source:https://user-images.githubusercontent.com/91752852/142595844-c2ace164-3571-431d-8e5c-e230eac7c1ca.gif)

- Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression.
- The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

Advantages and Disadvantages of Decision Trees

Some advantages of decision trees are:

- They are simple to understand, interpret and visualize.

- They require little data preparation. Any missing value present in the data does not affect a decision tree. So, it is considered a flexible algorithm.
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
- They can handle both numerical and categorical data. They can also handle multi-output problems.
- They use a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.
- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.
- They can perform well even if its assumptions are somewhat violated by the true model from which the data were generated.

Disadvantages of Decision tree

- Decision-tree learners can create over-complex trees that do not generalize the data well which is called overfitting.
- A decision tree is sometimes unstable and cannot be reliable as alteration in data can cause a decision tree to go in a bad structure which may affect the accuracy of the model.
- Predictions of decision trees are neither smooth nor continuous, but piecewise constant approximations. Therefore, they are not good at extrapolation.
- Greedy algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees, where the features and samples are randomly sampled with replacement.

- There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.
- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

Decision Tree structure

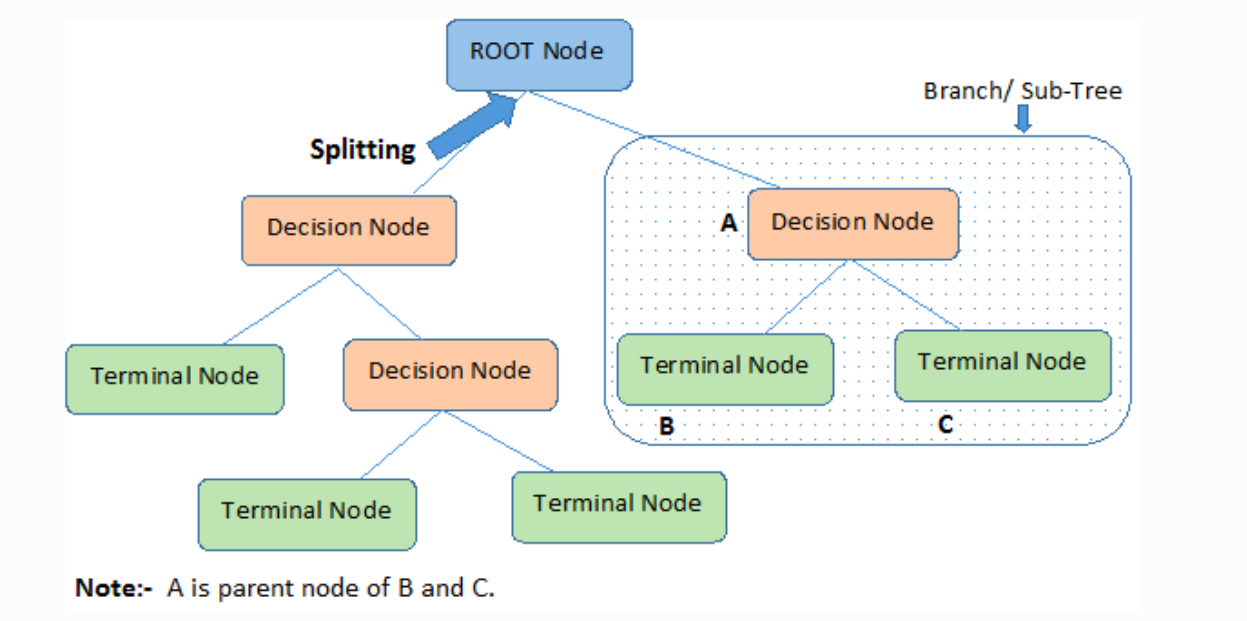


Fig 2: Structure of a Decision Tree(Source:laptrinhx.com)

Decision Tree Terminology

- Root node: Represents entire population
- Splitting: Process of dividing sample
- Decision Node: Node splits into further sub nodes
- Leaf / Terminal Node: Last stage of node (output label)
- Pruning: Opposite to splitting (to reduce size of tree)
- Parent and Child Node: A node, which is divided into sub-nodes is called parent node of sub-nodes whereas sub-nodes are the child of parent node.

- Branch / Sub-Tree: A subsection of the entire tree is called branch or sub-tree.

Types of Decision Tree

There are two type of Decision trees: Classification tree and Regression trees

- Classification trees help you classify categorical data for example loan status (approved/not approved), spam/not spam etc.
- Regression trees help you predict outcomes which can be a real number for example income of a person, sale price of a house etc.

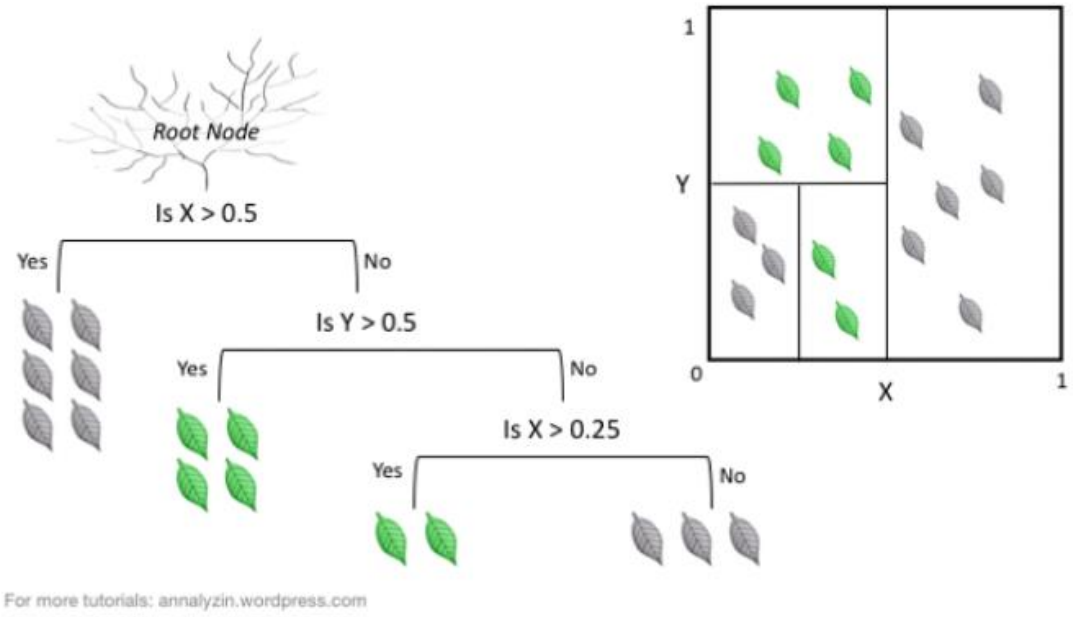


Fig 3: Classification and Regression Example

There are couple of algorithms there to build a decision tree. Some of them are as follows

- CART (Classification and Regression Trees) → uses Gini Index (Classification) as metric.
- ID3 (Iterative Dichotomiser 3) → uses Entropy function and Information gain as metrics.

Gini Index

- Gini Index is a metric to measure how often a randomly chosen element would be incorrectly identified.
- An attribute with lower Gini Index should be preferred.
- Mathematically:

$$\text{Gini Index} = 1-[(P)^2+(1-P)^2]$$

where P is the proportion of positive samples in that node. The Gini Index of '0' indicates that node is pure. Thus, it indicates no further splitting is needed.

Entropy

- It is defined as a measure of impurity present in the data.
- Entropy tends to be maximum in the middle with value up to 1 and minimum at the ends with value up to 0.
- Entropy with the lowest value makes a model better in terms of prediction as it separates the classes better.
- Mathematically,
  - Entropy = -p*log(p)* - *q*log(q)

Information Gain

- It is a measure used to generalize the impurity which is entropy in a dataset.
- In a decision tree building process, two important decisions are to be made: what is the best split(s) and which is the best variable to split a node.
- Higher the information gain, lower is the entropy.
- Mathematically,

  Information Gain = Entropy of Parent – sum (weighted % * Entropy of Child), Weighted % = Number of observations in particular child/sum (observations in all child nodes)

Chi Square

- Helps to find out the statistical significance between the differences among sub-nodes and parent nodes.
- We measure it by the sum of squares of standardized differences between observed and expected frequencies of the target variable.
- Mathematically represented as

$$\text{Chi-square} = ((\text{Actual} - \text{Expected})^2 / \text{Expected})^{1}/2$$

- It is a measure of purity. higher the value of Chi-square will be the higher in statistical significance of differences between sub-node and Parent node.

Reduction in Variance

All the methods mentioned above are applied to classification decision trees.

- In the case of a regression decision tree where the target variable is continuous, reduction in variance method is followed.
- It uses the standard formula of variance to choose the best split.
- The split with lower variance is selected as the criteria to split the population.
- Mathematically represented as

$$\text{Variance} = \frac{\Sigma(X - \overline{X})^2}{n}$$

Where X bar is the mean of values, X is the actual mean and n is the number of values.

How to solve Overfitting in a Decision Tree?

Overfitting is the main challenge in decision tree. Overfitting can be avoided by two methods.

Pruning

- Pruning is a technique in machine learning that reduces the size of decision trees by removing branches of the tree that are less important to classify instances.
- Pruning reduces the complexity of the final classifier and hence improves predictive accuracy by the reduction of over-fitting.
- There are mainly two methods of pruning
    - Pre-pruning – Growing tree is stopped earlier. It means we can prune/remove/cut a node if it has low importance while growing the tree.
    - Post-pruning – In post pruning, once our tree is built to its depth, we can start pruning the nodes based on their significance.

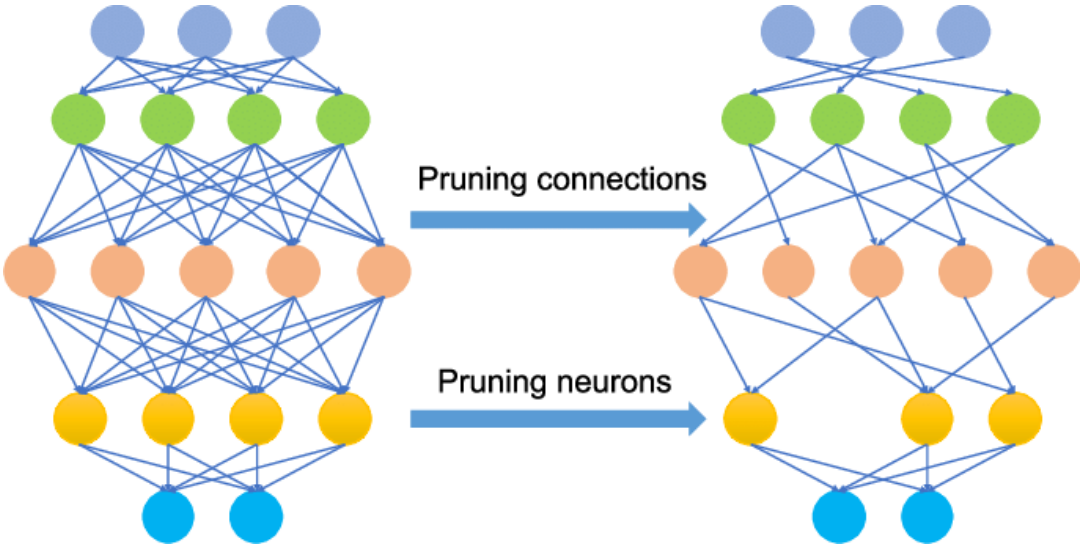
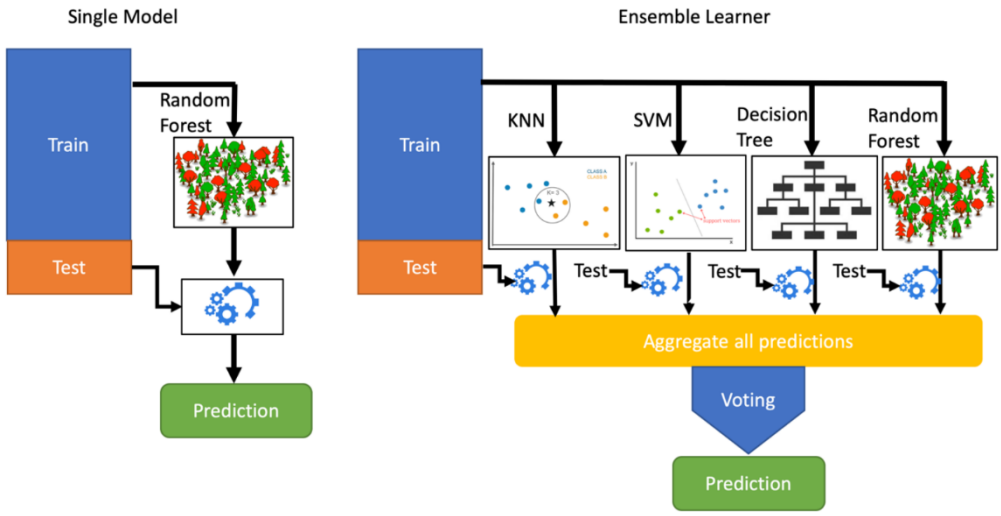


Fig 4 : Pruning a neural network(Source:[Deep Learning With Edge Computing: A Review(https://www.researchgate.net/publication/334489669_Deep_Learning_With_Edge_Computing_A_Review))

Ensemble method or bagging and boosting

- Ensemble learning is a general meta approach to machine learning that search for better predictive performance by combining the predictions from multiple models. There are main classes of ensemble learning methods are bagging, stacking, and boosting. Among them Boosting is a set of algorithms that focuses on converting weak learners to strong learner.



.

Fig 5 Single Model Prediction vs Ensemble Learner(Source:dzone.com)

BOOSTING

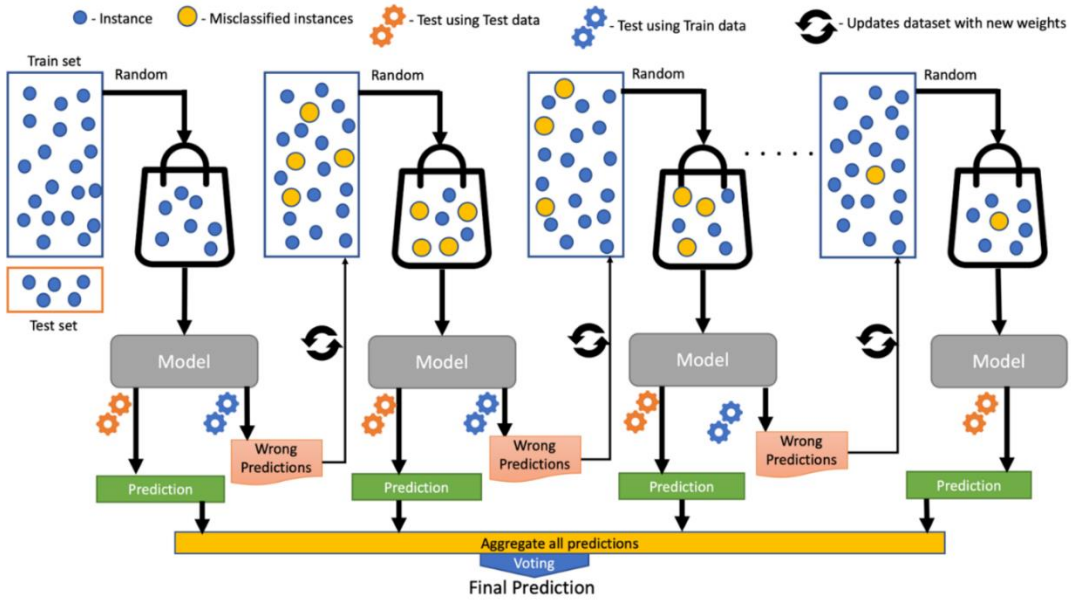


Fig 6: Internal working of boosting algorithm (Source:dzone.com)

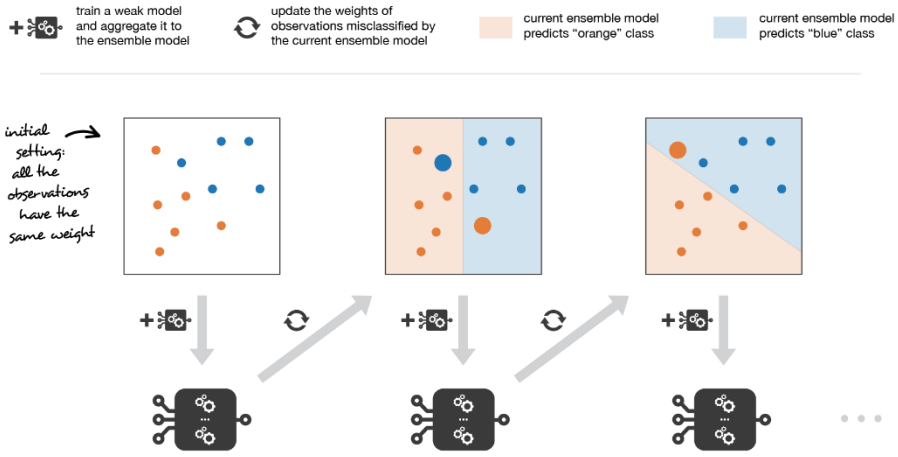
- Boosting was first introduced by Freund and Schapire in the year 1997 with their AdaBoost algorithm.

" *Boosting refers to a general and provably effective method of producing a very accurate prediction rule by combining rough and moderately inaccurate rules of thumb." ” (Schapire and Freund [2012])*

Types of Boosting

Adaboost:

- AdaBoost is implemented by combining several weak learners into a single strong learner.
- It was proposed by Yoav Freund and Robert Schapire.
- It operates iteratively, identifying misclassified data points and adjusting their weights to minimize the training error.
- The model continues to optimize in a sequential fashion until it yields the strongest predictor.



.

Fig 7: Adaboost updates weights of the observations at each iteration. Weights of well classified observations decrease relatively to weights of misclassified observations. Models that perform better have higher weights in the final ensemble model(Source:Ensemble methods: bagging, boosting and stacking)

Gradient Boosting (GBM):

- The name gradient boosting is derived from the combination of the gradient descent algorithm and boosting method.

- It was developed by Leo Breiman and Jerome H. Friedman.
- It works by sequentially adding predictors to an ensemble with each one correcting for the errors of its predecessor.
- However, instead of changing weights of data points like AdaBoost, the gradient boosting trains on the residual errors of the previous predictor.
- The main objective is to overcome the errors in the previous learner's predictions. This type of boosting has three main parts:
    - Loss function that needs to be ameliorated.
    - Weak learner for computing predictions and forming strong learners.
    - An Additive Model that will regularize the loss function.

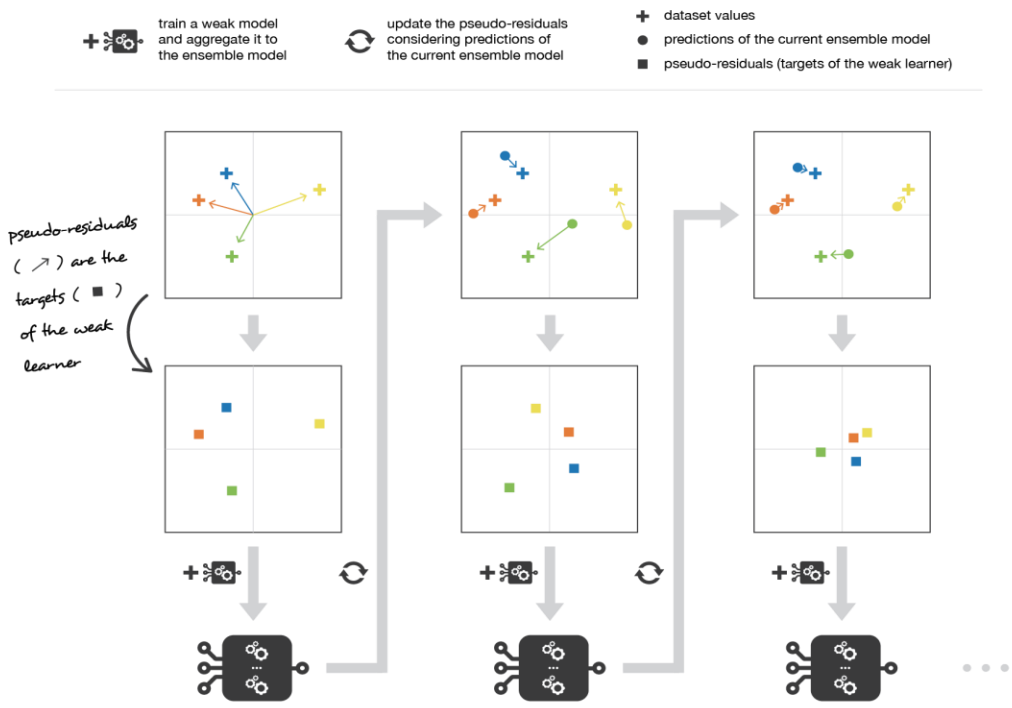


Fig 8:Gradient boosting updates values of the observations at each iteration. Weak learners are trained to fit the pseudo-residuals that indicate in which direction to correct the current ensemble model predictions to lower the error(Source:Ensemble methods: bagging, boosting and stacking)

Extreme gradient boosting or XGBoost:

- XGBoost is an implementation of gradient boosting that's designed for computational speed and scale.
- XGBoost leverages multiple cores on the CPU, allowing for learning to occur in parallel during training.

XGBOOST

- The extended form of XGBoost is Extreme Gradient Boosting.
- XGBoost uses a supervised learning algorithm to accurately predict a target variable by combining an ensemble of estimates from a set of simpler and weaker models.

- 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 is an extension to gradient boosted decision trees (GBM) and specially purposed to improve speed and performance.
- The XGBoost algorithm performs well in machine learning competitions because of its robust handling of a variety of data types, relationships, distributions, and the variety of hyperparameters that can be fine-tuned.
- XGBoost is used for regression, classification (binary and multiclass), and ranking problems.

Origin of XGBOOST

- The term "Gradient Boosting" was coined by Jerome H. Friedman from his paper Greedy Function Approximation: A Gradient Boosting Machine.
- XGBoost algorithm was developed by Tianqi Chen and Carlos Guestrin as a research project at the University of Washington at SIGKDD Conference in 2016.

Brief illustration on how gradient tree boosting works

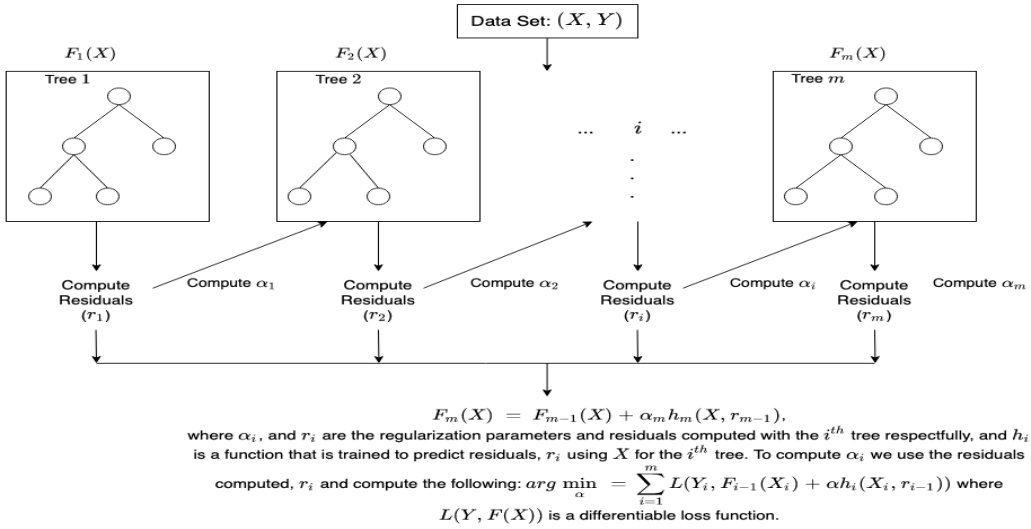


Fig 9: Brief illustration on how gradient tree boosting works Source:Amazon SageMaker Developer Guide

XGBoost Features

Fig 10: Features in XGBoost for optimization (Source: Comparative Analysis of Artificial Neural Network and XGBoost Algorithm for PolSAR Image Classification)

Regularized Learning:

- It penalizes more complex models through both LASSO (L1) and Ridge (L2) regularization to prevent overfitting.
- Smooth the final learnt weights to avoid over-fitting.
- The regularized objective is to select a model employing simple and predictive functions.

Gradient Tree Boosting:

- The tree ensemble model cannot be optimized using traditional optimization methods in Euclidean space.
- Instead, the model is trained in an additive manner.

Shrinkage and Column Subsampling:

- Two additional techniques Shrinkage and Column Subsampling are used after regularization to further prevent overfitting.

Shrinkage Subsampling

- Shrinkage scales newly added weights by a factor η after each step of tree boosting.
- Similar to a learning rate in stochastic optimization, shrinkage reduces the influence of each tree and leaves space for future trees to improve the model.

Column Subsampling

- Column (feature) subsampling uses Random Forest.
- Column sub-sampling prevents over-fitting even more so than the traditional row sub-sampling.
- Column sub-samples also boost up computations of the parallel algorithm.
- According to user feedback, using column sub-sampling prevents over-fitting even more so than the traditional row sub-sampling (which is also supported).

SPLITTING ALGORITHMS

Exact Greedy Algorithm:

- The main problem in tree learning is to find the best split.
- This algorithm enumerates over all the possible splits on all the features.
- It is computationally demanding to enumerate all the possible splits for continuous features.
- In order to do so efficiently, the algorithm must first sort the data according to feature values and visit the data in sorted order to accumulate the gradient statistics for the structure score .
- A split finding algorithm enumerates over all the possible splits on all the features. We call this the exact greedy algorithm.
- Most existing single machine tree boosting implementations, such as scikit-learn, R's gbm as well as the single machine version of XGBoost support the exact greedy algorithm.

Approximate Algorithm:

- The exact greedy algorithm enumerates over all possible splitting points greedily but it is not possible when the data does not fit entirely into memory.
- It proposes candidate splitting points according to percentiles of feature distribution.
- The algorithm then plans the continuous features into buckets split by these candidate points, aggregates the statistics and searches for the best solution among proposals based on the aggregated statistics.
- To support effective gradient tree boosting in these two settings, an approximate algorithm is needed.
- To summarize, the algorithm first proposes candidate splitting points according to percentiles of feature distribution. The algorithm then maps the continuous features into buckets split by these candidate points, aggregates the statistics and finds the best solution among proposals based on the aggregated statistics.
- There are two variants of the algorithm based on the proposal passed .The global variant proposes all the candidate splits during the initial phase of tree construction, and uses the same proposals for split finding at all levels. The local variant re-proposes after each split. The global method requires less proposal steps than the local method.
- However, usually more candidate points are needed for the global proposal because candidates are not refined after each split. The local proposal refines the candidates after splits, and can potentially be more appropriate for deeper trees. A comparison of different algorithms on a Higgs boson dataset. We find that the local proposal indeed requires fewer candidates. The global proposal can be as accurate as the local one given enough candidates.
- Most existing approximate algorithms for distributed tree learning also follow this framework. Notably, it is also possible to directly construct approximate histograms of gradient statistics.
- We can also use other variants of binning strategies instead of quantiles. Quantile strategy benefits from being distributable and recomputable, which will detail in next subsection that the quantile strategy can get the same accuracy as exact greedy given reasonable approximation level.
- Our system efficiently supports exact greedy for the single machine setting, as well as an approximate algorithm with both local and global

proposal methods for all settings. Users can freely choose between the methods according to their necessities.

Weighted Quantile Sketch:

- One important theme in the approximate algorithm is to propose candidate split points.
- XGBoost has a distributed weighted quantile sketch algorithm to effectively handle weighted data.
- For large datasets, it is non-trivial to find candidate splits that satisfy the criteria.
- When every instance has equal weights, an existing algorithm called quantile sketch solves the problem.
- However, there is no existing quantile sketch for the weighted datasets.
- Since, most existing approximate algorithms either resort to sorting on a random subset of data which have a chance of failure or heuristics that do not have theoretical guarantee.
- A novel distributed weighted quantile sketch algorithm is used 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.

Sparsity-aware Split Finding:

- XGBoost obviously admits sparse features for inputs by automatically 'learning' best missing value depending on training loss and handles all sparsity patterns in a unified way.
- In many real-world problems, it is quite common for the input x to be sparse. 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. 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. The key improvement is to only visit the non-missing entries.

- The presented algorithm treats the non-presence as a missing value and learns the best direction to handle missing values. The same algorithm can also be applied when the non-presence corresponds to a user specified value by limiting the enumeration only to consistent solutions.
- Most existing tree learning algorithms are either only optimized for dense data, or need specific procedures to handle limited cases such as categorical encoding.
- XGBoost handles all sparsity patterns in a unified way. More importantly, our method exploits the sparsity to make computation complexity linear to the number of non-missing entries in the input.
- The sparsity aware algorithm runs 50 times faster than the naive version which aware about its value as shown in figure

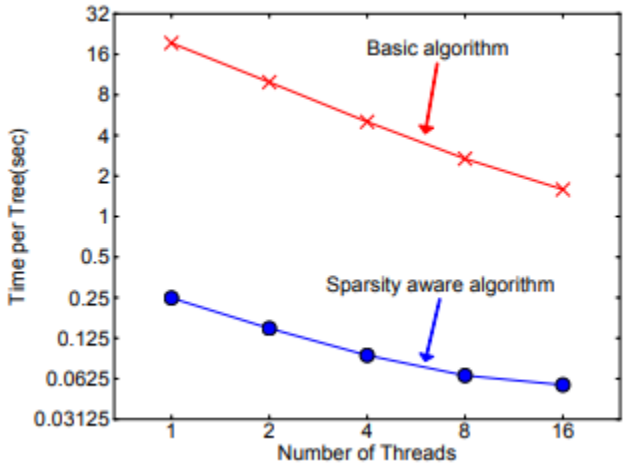


Fig 11: Impact of the sparsity aware algorithm on Allstate-10K.(Source: XGBoost: A Scalable Tree Boosting System)

System Optimization

The library provides a system for use in a range of computing environments.

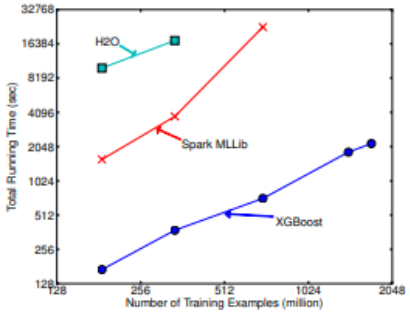
Parallelization

Parallelization of tree construction using all of your CPU cores during training.

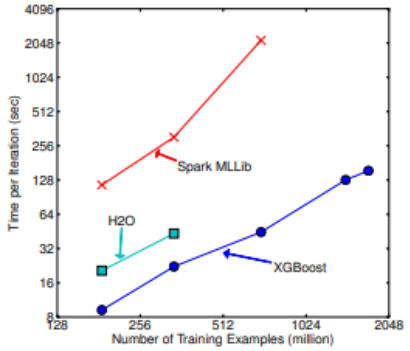
- XGBoost follows the process of sequential tree building using parallelized implementation.
- Because of the interchangeable nature of loops used for building base learners; the outer loop that enumerates the leaf nodes of a tree, and the second inner loop that calculates the features.
- This nesting 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

Distributed Computing

Distributed Computing for training very large models using a cluster of machines. XGBoost is able to handle the entire 1.7 billion data with only four machines. This shows the system's potential to handle even larger data.



(a) End-to-end time cost include data loading



(b) Per iteration cost exclude data loading

Fig 12: Comparison of different distributed systems on 32 EC2 nodes for 10 iterations on different subset of criteo data (Source: XGBoost: A Scalable Tree Boosting System)

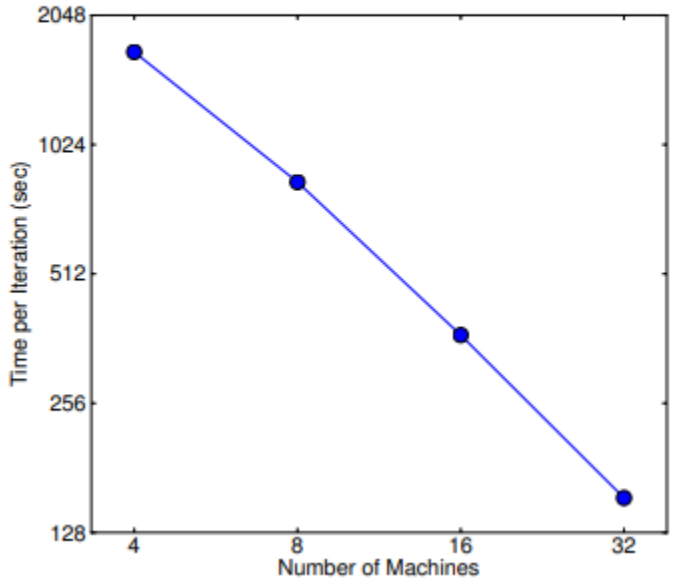


Fig 13 :Scaling of XGBoost with different number of machines on criteo full 1.7 billion dataset(Source: XGBoost: A Scalable Tree Boosting System)

Out-of-Core Computing

- It optimizes available disk space while handling big data-frames that do not fit into memory.
- One goal of the system is to fully utilize a machine's resources to achieve scalable learning. Besides processors and memory, it is important to utilize disk space to handle data that does not fit into main memory.
- To enable out-of-core computation, the data is divided into multiple blocks and stored each block on disk.
- During computation, it is important to use an independent thread to pre-fetch the block into a main memory buffer, so computation can happen in concurrence with disk reading. However, this does not

entirely solve the problem since the disk reading takes most of the computation time.

- It is important to reduce the overhead and increase the throughput of disk IO. Two techniques are mainly used to improve out-of-core computation.

Block Compression

- The first method is block compression. The block is compressed by columns, and decompressed on the fly by an independent thread when loading into main memory. This helps to trade some of the computation in decompression with the disk reading cost.
- We use a general-purpose compression algorithm for compressing the features values. For the row index, we subtract the row index by the beginning index of the block and use a 16bit integer to store each offset.
- This requires 216 examples per block, which is confirmed to be a good setting. In most of the dataset we tested, we achieve roughly a 26% to 29% compression ratio.

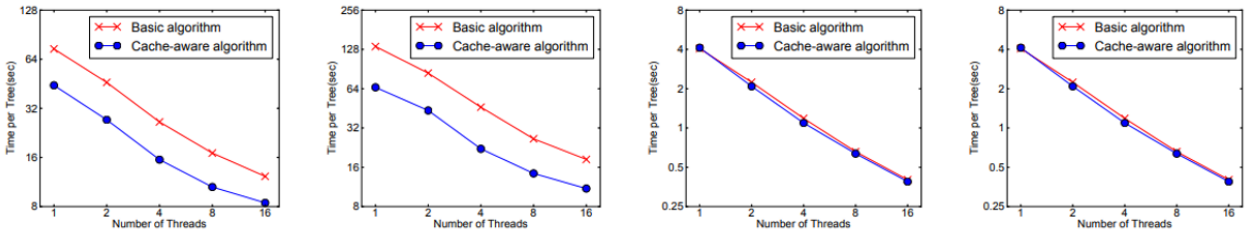
Block Sharding

- The second technique is to shard the data onto multiple disks in an alternative manner. A pre-fetcher thread is assigned to each disk and fetches the data into an in-memory buffer.
- The training thread then alternatively reads the data from each buffer which helps to increase the throughput of disk reading when multiple disks are available.

Cache Optimization

- Data structures and algorithms to make best use of hardware to store gradient statistics.
- While the proposed block structure helps optimize the computation complexity of split finding, the new algorithm requires indirect fetches of gradient statistics by row index, since these values are accessed in order of feature. This is a non-contiguous memory access.
- A naive implementation of split enumeration introduces immediate read/write dependency between the accumulation and the non-contiguous memory fetch operation.

- It slows down split finding when the gradient statistics do not fit into CPU cache and cache miss occurs. For the exact greedy algorithm, we can alleviate the problem by a cache-aware prefetching algorithm.
- Specifically, we allocate an internal buffer in each thread, fetch the gradient statistics into it, and then perform accumulation in a mini-batch manner.
- This prefetching changes the direct read/write dependency to a longer dependency and it helps to reduce the runtime overhead when the number of rows in the is large.
- Cache-aware implementation of the exact greedy algorithm runs twice as fast as the naive version when the dataset is large.
- For approximate algorithms, we solve the problem by choosing a correct block size. We define the block size to be the maximum number of examples in contained in a block, as this reflects the cache storage cost of gradient statistics.
- Choosing an overly small block size results in a small workload for each thread and leads to inefficient parallelization.
- On the other hand, overly large blocks result in cache misses, as the gradient statistics do not fit into the CPU cache. A good choice of block size balances these two factors.



(a) Allstate 10M (b) Higgs 10M (c) Allstate 1M (d) Higgs 1M

Fig 14: Impact of cache-aware prefetching in exact greedy algorithm (Source: XGBoost: A Scalable Tree Boosting System

Block structure for Parallel Learning:

- XGBoost can make use of multiple cores on the CPU for faster computing.
- This is due to block architecture in its system design.
- Data is sorted and stored in in-memory units called blocks.

- Different blocks can be distributed across machines, or stored on disk in the out-of-core setting.
- The block structure also helps when using the approximate algorithms.
- Using the sorted structure, the quantile finding step becomes a linear scan over the sorted columns. This is especially valuable for local proposal algorithms, where candidates are generated frequently at each branch.
- Column block structure also supports column subsampling, as it is easy to select a subset of columns in a block.

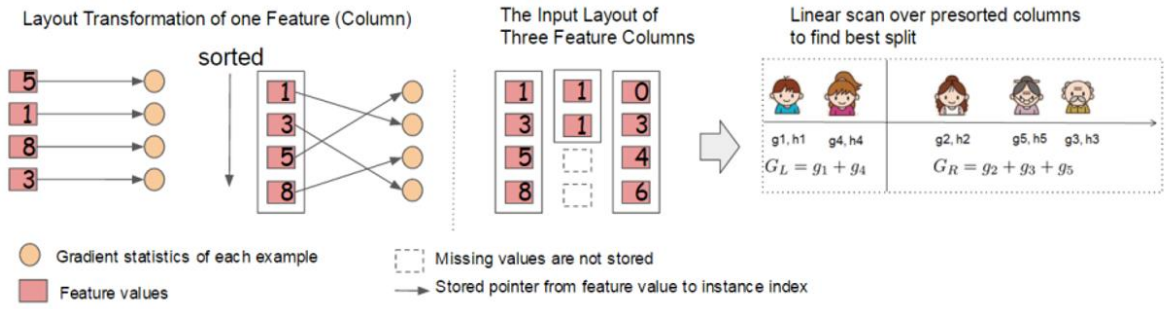


Fig 15:Block structure for parallel learning(Source: XGBoost)

Goal of XGBOOST

Execution Speed:

- XGBoost is almost always faster than the other benchmarked implementations from R, Python Spark and H2O
- It is really 10 times faster when compared to the other algorithms.

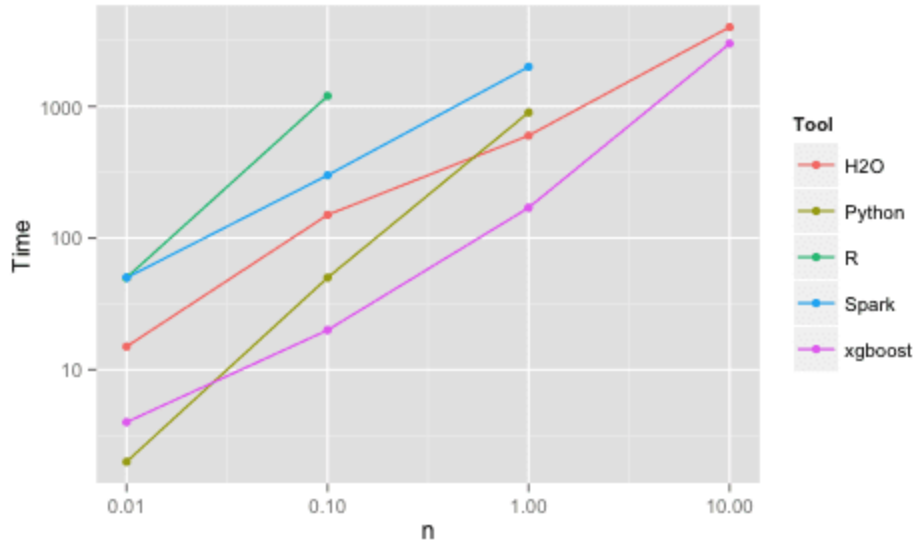


Fig 16: Benchmark Performance of XGBoost Source: Benchmarking Random Forest Implementations.

High Model Performance:

- XGBoost dominates structured or tabular datasets on classification and regression predictive modelling problems.

System Implementation

XGBoost is an open source package which is portable and reusable.

- XGBOOST has various applications to solve problems such as regression, classification, ranking, and user-defined prediction problems.
- Feasible to run smoothly on Windows, Linux, and OS X.
- Supports all popular programming languages such as C++, Python, R, Java, Scala, and Julia and integrates naturally with language native data science pipelines such as scikit learn.
- Cloud Integration such as AWS, Azure, Tianchi and Yarn clusters and works well with Flink, Spark, Hadoop, MPI Sun Grid engine and other ecosystems.

XGBoost Parameters

Three types of parameters in XGBoost: general parameters, booster parameters and task parameters as mentioned below.

General parameters are used for boosting, commonly tree or linear model

Booster parameters depend on user's preference

Learning task parameters decide on the learning scenario. For example, regression tasks may use different parameters with ranking tasks.

Command line parameters relate to behavior of CLI version of XGBoost.

How to Use XGBoost on Sage Maker?

We can use XGBoost in SageMaker in **built-in algorithms** or **frameworks**.

Use XGBoost as a framework

In the following example, Python SDK provides the XGBoost API as a framework while it provides other framework APIs, such as TensorFlow, MXNet, and PyTorch.

SageMaker Python SDK 1

```
import boto3
import sagemaker
from sagemaker.xgboost.estimator import XGBoost
from sagemaker.session import s3_input, Session

# initialize hyperparameters
hyperparameters = {
        "max_depth":"5",
        "eta":"0.2",
        "gamma":"4",
        "min_child_weight":"6",
        "subsample":"0.7",
        "verbosity":"1",
        "objective":"reg:linear",
        "num_round":"50"}

# set an output path where the trained model will be saved
bucket = sagemaker.Session().default_bucket()
prefix = 'DEMO-xgboost-as-a-framework'
output_path = 's3://{}/{}/{}/output'.format(bucket, prefix, 'abalone-xgb-framework')

# construct a SageMaker XGBoost estimator
# specify the entry_point to your xgboost training script
estimator = XGBoost(entry_point = "your_xgboost_abalone_script.py",
                    framework_version='1.2-2',
                    hyperparameters=hyperparameters,
                    role=sagemaker.get_execution_role(),
                    instance_count=1,
                    instance_type='ml.m5.2xlarge',
                    output_path=output_path)

# define the data type and paths to the training and validation datasets
content_type = "libsvm"
train_input = s3_input("s3://{}/{}/{}/".format(bucket, prefix, 'train'), content_type=content_type)
validation_input = s3_input("s3://{}/{}/{}/".format(bucket, prefix, 'validation'), content_type=content_type)

# execute the XGBoost training job
estimator.fit({'train': train_input, 'validation': validation_input})
```

SageMaker Python SDK 2

```
import boto3
import sagemaker
from sagemaker.xgboost.estimator import XGBoost
from sagemaker.session import Session
from sagemaker.inputs import TrainingInput

# initialize hyperparameters
hyperparameters = {
        "max_depth":"5",
        "eta":"0.2",
        "gamma":"4",
        "min_child_weight":"6",
        "subsample":"0.7",
        "verbosity":"1",
        "objective":"reg:linear",
        "num_round":"50"}

# set an output path where the trained model will be saved
bucket = sagemaker.Session().default_bucket()
prefix = 'DEMO-xgboost-as-a-framework'
output_path = 's3://{}/{}/{}/output'.format(bucket, prefix, 'abalone-xgb-framework')

# construct a SageMaker XGBoost estimator
# specify the entry_point to your xgboost training script
estimator = XGBoost(entry_point = "your_xgboost_abalone_script.py",
                    framework_version='1.2-2',
                    hyperparameters=hyperparameters,
                    role=sagemaker.get_execution_role(),
                    instance_count=1,
                    instance_type='ml.m5.2xlarge',
                    output_path=output_path)

# define the data type and paths to the training and validation datasets
content_type = "libsvm"
train_input = TrainingInput("s3://{}/{}/{}/".format(bucket, prefix, 'train'), content_type=content_type)
validation_input = TrainingInput("s3://{}/{}/{}/".format(bucket, prefix, 'validation'), content_type=content_type)

# execute the XGBoost training job
estimator.fit({'train': train_input, 'validation': validation_input})
```

Use XGBoost as a built-in algorithm

Amazon Sage Maker uses the XGBoost built-in algorithm to build an XGBoost training container as shown in the following code example.

SageMaker Python SDK 1

```
import sagemaker
import boto3
from sagemaker.amazon.amazon_estimator import get_image_uri
from sagemaker.session import s3_input, Session

# initialize hyperparameters
hyperparameters = {
        "max_depth":"5",
        "eta":"0.2",
        "gamma":"4",
        "min_child_weight":"6",
        "subsample":"0.7",
        "objective":"reg:squarederror",
        "num_round":"50"}

# set an output path where the trained model will be saved
bucket = sagemaker.Session().default_bucket()
prefix = 'DEMO-xgboost-as-a-built-in-algo'
output_path = 's3://{}/{}/{}/output'.format(bucket, prefix, 'abalone-xgb-built-in-algo')

# this line automatically looks for the XGBoost image URI and builds an XGBoost container.
# specify the repo_version depending on your preference.
xgboost_container = get_image_uri(boto3.Session().region_name,
                                  'xgboost',
                                  repo_version='1.2-2')

# construct a SageMaker estimator that calls the xgboost-container
estimator = sagemaker.estimator.Estimator(image_name=xgboost_container,
                                          hyperparameters=hyperparameters,
                                          role=sagemaker.get_execution_role(),
                                          instance_count=1,
                                          instance_type='ml.m5.2xlarge',
                                          train_volume_size=5, # 5 GB
                                          output_path=output_path)

# define the data type and paths to the training and validation datasets
content_type = "libsvm"
train_input = s3_input("s3://{}/{}/{}/".format(bucket, prefix, 'train'), content_type=content_type)
validation_input = s3_input("s3://{}/{}/{}/".format(bucket, prefix, 'validation'), content_type=content_type)

# execute the XGBoost training job
estimator.fit({'train': train_input, 'validation': validation_input})
```

SageMaker Python SDK 2

```
import sagemaker
import boto3
from sagemaker import image_uris
from sagemaker.session import Session
from sagemaker.inputs import TrainingInput

# initialize hyperparameters
hyperparameters = {
        "max_depth":"5",
        "eta":"0.2",
        "gamma":"4",
        "min_child_weight":"6",
        "subsample":"0.7",
        "objective":"reg:squarederror",
        "num_round":"50"}

# set an output path where the trained model will be saved
bucket = sagemaker.Session().default_bucket()
prefix = 'DEMO-xgboost-as-a-built-in-algo'
output_path = 's3://{}/{}/{}/output'.format(bucket, prefix, 'abalone-xgb-built-in-algo')

# this line automatically looks for the XGBoost image URI and builds an XGBoost container.
# specify the repo_version depending on your preference.
xgboost_container = sagemaker.image_uris.retrieve("xgboost", region, "1.2-2")

# construct a SageMaker estimator that calls the xgboost-container
estimator = sagemaker.estimator.Estimator(image_uri=xgboost_container,
                                          hyperparameters=hyperparameters,
                                          role=sagemaker.get_execution_role(),
                                          instance_count=1,
                                          instance_type='ml.m5.2xlarge',
                                          volume_size=5, # 5 GB
                                          output_path=output_path)

# define the data type and paths to the training and validation datasets
content_type = "libsvm"
train_input = TrainingInput("s3://{}/{}/{}/".format(bucket, prefix, 'train'), content_type=content_type)
validation_input = TrainingInput("s3://{}/{}/{}/".format(bucket, prefix, 'validation'), content_type=content_type)

# execute the XGBoost training job
estimator.fit({'train': train_input, 'validation': validation_input})
```

Regression with Amazon Sage Maker XGBoost algorithm

Introduction

This project presents the use of Amazon Sage Maker XGBoost to train and host a regression model.

We use the Abalone data, originally from the UCI data repository.

Here, the nominal feature(Male/Female/Infant) has been converted into a real valued feature as required by XGBoost. Age of abalone is to be predicted from eight physical measurements.

Set Up

This project was tested in Amazon SageMaker Studio on a ml.t3.medium instance with Python 3 kernel.

First specify the The S3 bucket and prefix that we want to use for training and model data. It should be within the same region as the Notebook Instance, training, and hosting.

The IAM roles are used to give training and hosting access to our data.

```
%%time

import os
import boto3
import re
import sagemaker

# Get a SageMaker-compatible role used by this Notebook Instance.
role = sagemaker.get_execution_role()
region = boto3.Session().region_name

### update below values appropriately ###
bucket = sagemaker.Session().default_bucket()
prefix = "sagemaker/DEMO-xgboost-dist-script"
####

print(region)
```

Fetch Dataset

Split the data into train/test/validation datasets and upload files to S3 as shown below.

```
%%time

import io
import boto3
import random


def data_split(
    FILE_DATA,
    DATA_DIR,
    FILE_TRAIN_BASE,
    FILE_TRAIN_1,
    FILE_VALIDATION,
    FILE_TEST,
    PERCENT_TRAIN_0,
    PERCENT_TRAIN_1,
    PERCENT_VALIDATION,
    PERCENT_TEST,
):
    data = [l for l in open(FILE_DATA, "r")]
    train_file_0 = open(DATA_DIR + "/" + FILE_TRAIN_0, "w")
    train_file_1 = open(DATA_DIR + "/" + FILE_TRAIN_1, "w")
    valid_file = open(DATA_DIR + "/" + FILE_VALIDATION, "w")
    tests_file = open(DATA_DIR + "/" + FILE_TEST, "w")

    num_of_data = len(data)
    num_train_0 = int((PERCENT_TRAIN_0 / 100.0) * num_of_data)
    num_train_1 = int((PERCENT_TRAIN_1 / 100.0) * num_of_data)
    num_valid = int((PERCENT_VALIDATION / 100.0) * num_of_data)
    num_tests = int((PERCENT_TEST / 100.0) * num_of_data)

    data_fractions = [num_train_0, num_train_1, num_valid, num_tests]
    split_data = [[], [], [], []]

    rand_data_ind = 0

    for split_ind, fraction in enumerate(data_fractions):
        for i in range(fraction):
            rand_data_ind = random.randint(0, len(data) - 1)
            split_data[split_ind].append(data[rand_data_ind])
            data.pop(rand_data_ind)
```

```
    for l in split_data[0]:
        train_file_0.write(l)

    for l in split_data[1]:
        train_file_1.write(l)

    for l in split_data[2]:
        valid_file.write(l)

    for l in split_data[3]:
        tests_file.write(l)

    train_file_0.close()
    train_file_1.close()
    valid_file.close()
    tests_file.close()


def write_to_s3(fobj, bucket, key):
    return (
        boto3.Session(region_name=region)
        .resource("s3")
        .Bucket(bucket)
        .Object(key)
        .upload_fileobj(fobj)
    )


def upload_to_s3(bucket, channel, filename):
    fobj = open(filename, "rb")
    key = prefix + "/" + channel
    url = "s3://{}/{}/{}".format(bucket, key, filename)
    print("Writing to {}".format(url))
    write_to_s3(fobj, bucket, key)
```

Data ingestion

- Read the dataset from the existing repository into memory, for preprocessing prior to training.
- It can be done in situ by Amazon Athena, Apache Spark in Amazon EMR, Amazon Redshift, etc. We believe that the dataset is present in the right location.

- After that, we transfer the data to S3 for use in training. For small datasets, such as this one, reading into memory isn't onerous, though it would be for larger datasets.

```
%%time
s3 = boto3.client("s3")

# Load the dataset
FILE_DATA = "abalone"
s3.download_file(
    "sagemaker-sample-files", f"datasets/tabular/uci_abalone/abalone.libsvm", FILE_DATA
)

# split the downloaded data into train/test/validation files
FILE_TRAIN_0 = "abalone.train_0"
FILE_TRAIN_1 = "abalone.train_1"
FILE_VALIDATION = "abalone.validation"
FILE_TEST = "abalone.test"
PERCENT_TRAIN_0 = 35
PERCENT_TRAIN_1 = 35
PERCENT_VALIDATION = 15
PERCENT_TEST = 15

DATA_DIR = "data"

if not os.path.exists(DATA_DIR):
    os.mkdir(DATA_DIR)

data_split(
    FILE_DATA,
    DATA_DIR,
    FILE_TRAIN_0,
    FILE_TRAIN_1,
    FILE_VALIDATION,
    FILE_TEST,
    PERCENT_TRAIN_0,
    PERCENT_TRAIN_1,
    PERCENT_VALIDATION,
    PERCENT_TEST,
)
```

```
# upload the files to the S3 bucket
upload_to_s3(bucket, "train/train_0.libsvm", DATA_DIR + "/" + FILE_TRAIN_0)
upload_to_s3(bucket, "train/train_1.libsvm", DATA_DIR + "/" + FILE_TRAIN_1)
upload_to_s3(bucket, "validation/validation.libsvm", DATA_DIR + "/" + FILE_VALIDATION)
upload_to_s3(bucket, "test/test.libsvm", DATA_DIR + "/" + FILE_TEST)
```

Create a XGBoost script to train with

- SageMaker can now run an XGboost script using the XGBoost estimator.
- Two input channels, 'train' and 'validation', were used in the call to the XGBoost estimator's fit () method.
- A typical training script loads data from the input channels, configures training with hyperparameters, trains a model, and saves a model to model_dir so that it can be hosted later.
- Hyperparameters are passed to the script as arguments and can be retrieved with an argparse. Argument Parser instance.
- For instance, the script run in this notebook is provided as the accompanying file (abalone.py) and also shown below:

```
import argparse
import json
import logging
import os
import pandas as pd
import pickle as pkl

from sagemaker_containers import entry_point
from sagemaker_xgboost_container.data_utils import get_dmatrix
from sagemaker_xgboost_container import distributed

import xgboost as xgb


def _xgb_train(params, dtrain, evals, num_boost_round, model_dir, is_master):
    """Run xgb train on arguments given with rabit initialized.

    This is our rabit execution function.

    :param args_dict: Argument dictionary used to run xgb.train().
    :param is_master: True if current node is master host in distributed training,
                      or is running single node training job.
                      Note that rabit_run will include this argument.
    """
    booster = xgb.train(params=params,
                        dtrain=dtrain,
                        evals=evals,
                        num_boost_round=num_boost_round)

    if is_master:
        model_location = model_dir + '/xgboost-model'
        pkl.dump(booster, open(model_location, 'wb'))
        logging.info("Stored trained model at {}".format(model_location))


if __name__ == '__main__':
    parser = argparse.ArgumentParser()

    # Hyperparameters are described here.
```

```
# Hyperparameters are described here.
parser.add_argument('--max_depth', type=int,)
parser.add_argument('--eta', type=float)
parser.add_argument('--gamma', type=int)
parser.add_argument('--min_child_weight', type=int)
parser.add_argument('--subsample', type=float)
parser.add_argument('--verbosity', type=int)
parser.add_argument('--objective', type=str)
parser.add_argument('--num_round', type=int)
parser.add_argument('--tree_method', type=str, default="auto")
parser.add_argument('--predictor', type=str, default="auto")

# Sagemaker specific arguments. Defaults are set in the environment variables.
parser.add_argument('--output_data_dir', type=str, default=os.environ.get('SM_OUTPUT_DATA_DIR'))
parser.add_argument('--model_dir', type=str, default=os.environ.get('SM_MODEL_DIR'))
parser.add_argument('--train', type=str, default=os.environ.get('SM_CHANNEL_TRAIN'))
parser.add_argument('--validation', type=str, default=os.environ.get('SM_CHANNEL_VALIDATION'))
parser.add_argument('--sm_hosts', type=str, default=os.environ.get('SM_HOSTS'))
parser.add_argument('--sm_current_host', type=str, default=os.environ.get('SM_CURRENT_HOST'))

args, _ = parser.parse_known_args()

# Get SageMaker host information from runtime environment variables
sm_hosts = json.loads(args.sm_hosts)
sm_current_host = args.sm_current_host

dtrain = get_dmatrix(args.train, 'libsvm')
dval = get_dmatrix(args.validation, 'libsvm')
watchlist = [(dtrain, 'train'), (dval, 'validation')] if dval is not None else [(dtrain, 'train')]
```

```
train_hp = {
    'max_depth': args.max_depth,
    'eta': args.eta,
    'gamma': args.gamma,
    'min_child_weight': args.min_child_weight,
    'subsample': args.subsample,
    'verbosity': args.verbosity,
    'objective': args.objective,
    'tree_method': args.tree_method,
    'predictor': args.predictor,
}

xgb_train_args = dict(
    params=train_hp,
    dtrain=dtrain,
    evals=watchlist,
    num_boost_round=args.num_round,
    model_dir=args.model_dir)

if len(sm_hosts) > 1:
    # Wait until all hosts are able to find each other
    entry_point._wait_hostname_resolution()

    # Execute training function after initializing rabit.
    distributed.rabit_run(
        exec_fun=_xgb_train,
        args=xgb_train_args,
        include_in_training=(dtrain is not None),
        hosts=sm_hosts,
        current_host=sm_current_host,
        update_rabit_args=True
    )
else:
    # If single node training, call training method directly.
    if dtrain:
        xgb_train_args['is_master'] = True
        _xgb_train(**xgb_train_args)
    else:
        raise ValueError("Training channel must have data to train model.")
```

```
def model_fn(model_dir):
    """Deserialize and return fitted model.

    Note that this should have the same name as the serialized model in the _xgb_train method
    """
    model_file = 'xgboost-model'
    booster = pkl.load(open(os.path.join(model_dir, model_file), 'rb'))
    return booster
```

Because the container imports our training script, always put training code in a main guard [if **name**=='**main**':] so that the container does not inadvertently run our training code at the wrong point in execution.

Train the XGBoost model

- After setting training parameters, we start training, and poll for status until training is completed as shown below.

We construct a sagemaker.xgboost.estimator to run our training script on SageMaker

```
hyperparams = {
    "max_depth": "5",
    "eta": "0.2",
    "gamma": "4",
    "min_child_weight": "6",
    "subsample": "0.7",
    "objective": "reg:squarederror",
    "num_round": "50",
    "verbosity": "2",
}

instance_type = "ml.m5.2xlarge"
output_path = "s3://{}/{}/{}/output".format(bucket, prefix, "abalone-dist-xgb")
content_type = "libsvm"
```

```
  # Open Source distributed script mode
from sagemaker.session import Session
from sagemaker.inputs import TrainingInput
from sagemaker.xgboost.estimator import XGBoost

boto_session = boto3.Session(region_name=region)
session = Session(boto_session=boto_session)
script_path = "abalone.py"

xgb_script_mode_estimator = XGBoost(
    entry_point=script_path,
    framework_version="1.3-1",  # Note: framework_version is mandatory
    hyperparameters=hyperparams,
    role=role,
    instance_count=2,
    instance_type=instance_type,
    output_path=output_path,
)

train_input = TrainingInput(
    "s3://{}/{}/{}/".format(bucket, prefix, "train"), content_type=content_type
)
validation_input = TrainingInput(
    "s3://{}/{}/{}/".format(bucket, prefix, "validation"), content_type=content_type
)
```

Train XGBoost Estimator on abalone data

Training is as simple as calling fit on the Estimator. It starts a SageMaker Training job that will download the data, invoke the entry point code in the provided script file, and save any model artifacts that the script creates

```
xgb_script_mode_estimator.fit({"train": train_input, "validation": validation_input})
```

Deploy the XGBoost model

```
predictor = xgb_script_mode_estimator.deploy(
    initial_instance_count=1, instance_type="ml.m5.2xlarge"
)
```

```
test_file = DATA_DIR + "/" + FILE_TEST
with open(test_file, "r") as f:
    payload = f.read()
```

```
runtime_client = boto3.client("runtime.sagemaker", region_name=region)
response = runtime_client.invoke_endpoint(
    EndpointName=predictor.endpoint_name, ContentType="text/libsvm", Body=payload
)
result = response["Body"].read().decode("ascii")
print("Predicted values are {}.".format(result))
```

Delete the Endpoint

After completion, we can run the delete endpoint line in the cell as given. It will remove the hosted endpoint and avoid paying from a stray instance being left on.

```
predictor.delete_endpoint()
```

Research Papers on XGBOOST

- Forecasting gold price with the XGBoost algorithm and SHAP interaction values : The utilization of XGBoost and SHAP approach could pro- vide a significant boost in increasing the gold price forecasting performance.
- Model Research on Forecast of Second-Hand House Price in Chengdu Based on XGboost Algorithm : the accuracy of XGboost prediction is the highest with 0.9251 compared with linear regression and decision tree model, XGboost algorithm has better generalization ability and robustness in data prediction.

- SubMito-XGBoost: predicting protein submitochondrial localization by fusing multiple feature information and eXtreme gradient boosting The prediction accuracies of the SubMito-XGBoost method on the two training datasets M317 and M983 were 97.7% and 98.9%, which are 2.8–12.5% and 3.8–9.9% higher than other methods, respectively and also independent test set M495 was 94.8%, which is significantly better than the existing studies.
- Detection of Parkinson's Disease by Employing Boosting Algorithms (2021) : XGBoost and Gradient Boosting algorithm gave accuracies more than 90% in Parkinson's Disease detection
- Application of XGBoost Algorithm in The Detection of SARS-CoV-2 Using Raman Spectroscopy : XGBOOST detected a novel coronavirus with an accuracy of 93.548%.
- model: XGBoost is applicable for streamflow forecasting and performs better than SVM.

Reference

- Alon, Noga, Alon Gonen, Elad Hazan, and Shay Moran. "Boosting Simple Learners." ArXiv:2001.11704 [Cs, Stat], April 8, 2021. http://arxiv.org/abs/2001.11704.
- Aws/Sagemaker-Containers. Python. 2018. Reprint, Amazon Web Services, 2021. https://github.com/aws/sagemaker-containers.
- Analytics Vidhya. "Best Boosting Algorithm In Machine Learning In 2021," April 27, 2021. https://www.analyticsvidhya.com/blog/2021/04/best-boosting-algorithm-in-machine-learning-in-2021/.
- Bühlmann, Peter, and Torsten Hothorn. "Boosting Algorithms: Regularization, Prediction and Model Fitting." Statistical Science 22, no. 4 (November 1, 2007). https://doi.org/10.1214/07-STS242.
- Chen, Tianqi, and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System." Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, August 13, 2016, 785–94. https://doi.org/10.1145/2939672.2939785.
- "Docker Registry Paths and Example Code - Amazon SageMaker." Accessed November 18, 2021. https://docs.aws.amazon.com/sagemaker/latest/dg/sagemaker-algo-docker-registry-paths.html.
- EXtreme Gradient Boosting. C++. 2014. Reprint, Distributed (Deep) Machine Learning Community, 2021. https://github.com/dmlc/xgboost.
- Intergovernmental Panel on Climate Change, ed. "Summary for Policymakers." In Climate Change 2013 – The Physical Science Basis: Working Group I Contribution to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change, 1–30. Cambridge: Cambridge University Press, 2014. https://doi.org/10.1017/CBO9781107415324.004.
- Li, Cheng. "A Gentle Introduction to Gradient Boosting," n.d., 80.
- "Managed Spot Training for XGBoost — Amazon SageMaker Examples 1.0.0 Documentation." Accessed November 18, 2021. https://sagemaker-examples.readthedocs.io/en/latest/introduction_to_amazon_algorithms/xgboost_abalone/xgboost_managed_spot_training.html.

- Mayr, Andreas, Harald Binder, Olaf Gefeller, and Matthias Schmid. "The Evolution of Boosting Algorithms - From Machine Learning to Statistical Modelling." Methods of Information in Medicine 53, no. 06 (2014): 419–27. https://doi.org/10.3414/ME13-01-0122.
- Memon, Nimra, Samir Patel, and Dhruvesh Patel. "Comparative Analysis of Artificial Neural Network and XGBoost Algorithm for PolSAR Image Classification," 452–60, 2019. https://doi.org/10.1007/978-3-030-34869-4_49.
- Nishat, Mirza, Tasnimul Hasan, Sarker Nasrullah, Fahim Faisal, Md. Asfi Asif, and Md. Ashraful Hoque. Detection of Parkinson's Disease by Employing Boosting Algorithms, 2021. https://doi.org/10.1109/ICIEVicIVPR52578.2021.9564108.
- Pafka, Szilard. Szilard/Benchm-Ml. R, 2021. https://github.com/szilard/benchm-ml.
- "Parallel Gradient Boosting Decision Trees." Accessed November 18, 2021. http://zhanpengfang.github.io/418home.html.
- Parikh, Biraj. "Decision Tree Intuition." GreyAtom (blog), August 9, 2017. https://medium.com/greyatom/decision-tree-intuition-a38669005cb7.
- "Regression with Amazon SageMaker XGBoost Algorithm — Amazon SageMaker Examples 1.0.0 Documentation." Accessed November 18, 2021. https://sagemaker-examples.readthedocs.io/en/latest/introduction_to_amazon_algorithms/xgboost_abalone/xgboost_abalone_dist_script_mode.html.
- Rocca, Joseph. "Ensemble Methods: Bagging, Boosting and Stacking." Medium, March 21, 2021. https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205.
- "Story and Lessons Behind the Evolution of XGBoost - Nttrungmt-Wiki." Accessed November 18, 2021. https://sites.google.com/site/nttrungmtwiki/home/it/data-science---python/xgboost/story-and-lessons-behind-the-evolution-of-xgboost.
- Team, Great Learning. "Introduction to Decision Tree Algorithm - Explained with Examples." GreatLearning Blog: Free Resources what Matters to shape your Career!, February 13, 2020. https://www.mygreatlearning.com/blog/decision-tree-algorithm/.

- “Understanding XGBoost Algorithm | What Is XGBoost Algorithm?” GreatLearning Blog: Free Resources what Matters to shape your Career!, October 22, 2020. https://www.mygreatlearning.com/blog/xgboost-algorithm/.
- Analytics Vidhya. “Tree Based Algorithms | Implementation In Python & R,” April 11, 2016. https://www.analyticsvidhya.com/blog/2016/04/tree-based-algorithms-complete-tutorial-scratch-in-python/.
- “UCI Machine Learning Repository: Abalone Data Set.” Accessed November 18, 2021. https://archive.ics.uci.edu/ml/datasets/abalone.
- Technovert. “What Is Decision Tree - Technovert Solutions,” April 25, 2019. https://technovert.com/blog/what-is-decision-tree/.
- dzone.com. “XGBoost: A Deep Dive Into Boosting - DZone AI.” Accessed November 19, 2021. https://dzone.com/articles/xgboost-a-deep-dive-into-boosting.
- “XGBoost Algorithm - Amazon SageMaker.” Accessed November 18, 2021. https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost.html.
- “XGBoost Tutorials — Xgboost 1.6.0-Dev Documentation.” Accessed November 18, 2021. https://xgboost.readthedocs.io/en/latest/tutorials/index.html.