XGBoost

XGBoost stands for e**X**treme **G**radient **B**oosting.

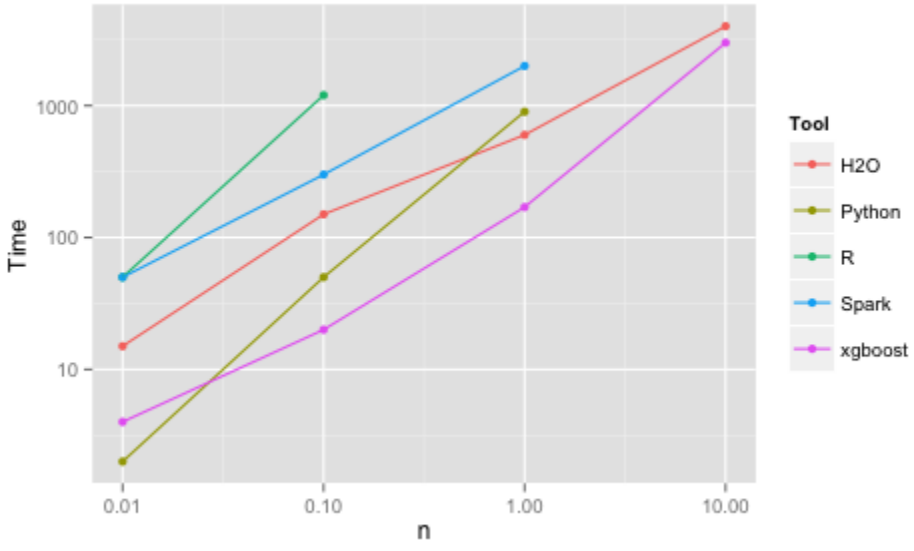
XGBoost is an algorithm that is an implementation of gradient boosted decision trees designed for speed and performance. It is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework.

For 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.

“The name XGBoost, though, actually refers to the engineering goal to push the limit of computation resources for boosted tree algorithms. Which is the reason why many people use XGBoost.” - Tianqi Chen, the co-creator of the XGBoost algorithm.

Why XGBoost?

- Execution speed: XGBoost is really fast, when compared to the other gradient boosting algorithms.



- Model performance: XGBoost dominates structured or tabular datasets on classification and regression predictive modeling problems.

The beauty of this powerful algorithm lies in its scalability, which drives fast learning through parallel and distributed computing and offers efficient memory usage. It is an ensemble learning method and it combines the power of multiple learners. The resultant is a single model which gives the aggregated output of several different models.

Features of XGBoost:

- Regularization: XGBoost has an option to penalize complex models through both L1 and L2 regularization which helps in preventing overfitting in the model.
- Handling sparse data: Missing values or data processing steps like one-hot encoding make data sparse. XGBoost incorporates a sparsity-aware split finding algorithm to handle different types of sparsity patterns in the data.
- Block structure for parallel learning: For faster computing, XGBoost can make use of multiple cores on the CPU. This is possible because of a block structure in its system design. Data is sorted and stored in in-memory units called blocks. Unlike other algorithms, this enables the data layout to be reused by subsequent iterations, instead of computing it again. This feature also serves useful for steps like split finding and column sub-sampling

The logic behind XGBoost:

Consider the following data where the years of experience is predictor variable and salary (in thousand dollars) is the target. Using regression trees as base learners, we can create a model to predict the salary. For the sake of simplicity, we can choose square loss as our loss function and our objective would be to minimize the square error.

Years	Salary
5	82
7	80
12	103
23	118
25	172
28	127
29	204
34	189
35	99
40	166

As the first step, the model should be initialized with a function F0(x). F0(x) should

be a function which minimizes the loss function or MSE (mean squared error), I

this case:

$$F_0(x) = argmin_\gamma \sum_{i=1}^{n} L(y_i, \gamma)$$

$$argmin_\gamma \sum_{i=1}^{n} L(y_i, \gamma) = argmin_\gamma \sum_{i=1}^{n} (y_i - \gamma)^2$$

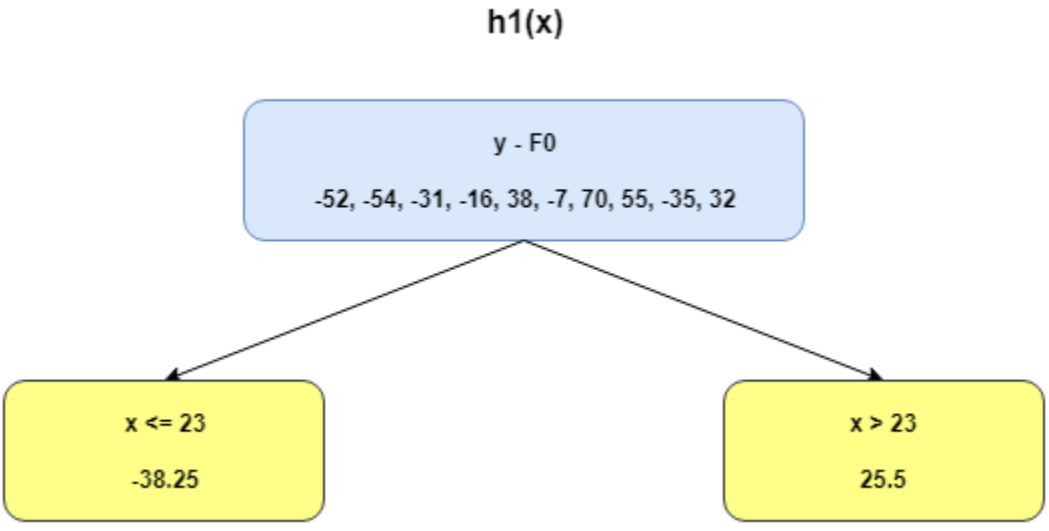
Taking the first differential of the above equation with respect to γ, it is seen that the function minimizes at the mean i=1nyin. So, the boosting model could be initiated with:

$$F_0(x) = \frac{\sum_{i=1}^{n} y_i}{n}$$

F0(x) gives the predictions from the first stage of our model. Now, the residual error for each instance is (yi – F0(x)).

x	y	F0	y - F0
5	82	134	-52
7	80	134	-54
12	103	134	-31
23	118	134	-16
25	172	134	38
28	127	134	-7
29	204	134	70
34	189	134	55
35	99	134	-35
40	166	134	32

We can use the residuals from F0(x) to create h1(x). h1(x) will be a regression tree which will try and reduce the residuals from the previous step. The output of h1(x) won't be a prediction of y; instead, it will help in predicting the successive function F1(x) which will bring down the residuals.



The additive model h1(x) computes the mean of the residuals (y – F0) at each leaf of the tree. The boosted function F1(x) is obtained by summing F0(x) and h1(x). This way h1(x) learns from the residuals of F0(x) and suppresses it in F1(x).

x	y	F0	y-F0	h1	F1
5	82	134	-52	-38.25	95.75
7	80	134	-54	-38.25	95.75
12	103	134	-31	-38.25	95.75
23	118	134	-16	-38.25	95.75
25	172	134	38	25.50	159.50
28	127	134	-7	25.50	159.50
29	204	134	70	25.50	159.50
34	189	134	55	25.50	159.50
35	99	134	-35	25.50	159.50
40	166	134	32	25.50	159.50

This can be repeated for 2 more iterations to compute h2(x) and h3(x). Each of these additive learners, hm(x), will make use of the residuals from the preceding function, Fm-1(x).

x	y	F0	y-F0	h1	F1	y-F1	h2	F2	y-F2	h3	F3
5	82	134	-52	-38.25	95.75	-13.75	6.75	102.50	-20.50	-10.08333	92.41667
7	80	134	-54	-38.25	95.75	-15.75	6.75	102.50	-22.50	-10.08333	92.41667
12	103	134	-31	-38.25	95.75	7.25	6.75	102.50	0.50	-10.08333	92.41667
23	118	134	-16	-38.25	95.75	22.25	6.75	102.50	15.50	-10.08333	92.41667
25	172	134	38	25.50	159.50	12.50	6.75	166.25	5.75	-10.08333	156.16667
28	127	134	-7	25.50	159.50	-32.50	6.75	166.25	-39.25	-10.08333	156.16667
29	204	134	70	25.50	159.50	44.50	6.75	166.25	37.75	15.12500	181.37500
34	189	134	55	25.50	159.50	29.50	6.75	166.25	22.75	15.12500	181.37500
35	99	134	-35	25.50	159.50	-60.50	-27.00	132.50	-33.50	15.12500	147.62500
40	166	134	32	25.50	159.50	6.50	-27.00	132.50	33.50	15.12500	147.62500

The MSEs for F0(x), F1(x) and F2(x) are 875, 692 and 540. It's amazing how these simple weak learners can bring about a huge reduction in error!

Pros of XGBoost:

- Extremely fast (parallel computation).
- Highly efficient.
- Versatile (Can be used for classification, regression or ranking).
- Can be used to extract variable importance.
- Do not require feature engineering (missing values imputation, scaling and normalization)

Cons of XGBoost:

- Only works with numeric features.
- Leads to overfitting if hyperparameters are not tuned properly.

Applications of XGBoost:

- Winning solution for many Kaggle competitions.
- Heavily used in industries due to its scalability.