**Introduction to XGBoost Algorithm in Machine Learning**

[**guest\_blog**](https://www.analyticsvidhya.com/blog/author/guest-blog/)**— Updated On March 30th, 2023**

[Algorithm](https://www.analyticsvidhya.com/blog/category/algorithm/) [Classification](https://www.analyticsvidhya.com/blog/category/classification/) [Intermediate](https://www.analyticsvidhya.com/blog/category/intermediate/) [Machine Learning](https://www.analyticsvidhya.com/blog/category/machine-learning/) [Python](https://www.analyticsvidhya.com/blog/category/python-2/) [Structured Data](https://www.analyticsvidhya.com/blog/category/structured-data/) [Supervised](https://www.analyticsvidhya.com/blog/category/supervised/)

Ever since its introduction in 2014, XGBoost has been lauded as the holy grail of [machine learning hackathons](https://www.analyticsvidhya.com/blog/2020/10/12-powerful-tips-ace-data-science-hackathons/) and competitions. From predicting ad click-through rates to classifying high energy physics events, XGBoost has proved its mettle in terms of performance – and speed. XGBoost is the first algorithm of choice for many in ML hackathon. The accuracy it consistently gives, and the time it saves, demonstrates how useful it is. But how does it actually work? What kind of mathematics power XGBoost? We’ll figure out the answers to these questions soon.

In this article, we will first look at the power of XGBoost, and then deep dive into the inner workings of this popular and powerful technique. It’s good to be able to implement it in [Python](https://www.analyticsvidhya.com/blog/2016/01/complete-tutorial-learn-data-science-python-scratch-2/) or [R](https://www.analyticsvidhya.com/blog/2022/02/r-programming-concepts-made-easy/), but understanding the nitty-gritties of the algorithm will help you become a better [data scientist.](https://blackbelt.analyticsvidhya.com/plus)

Table of contents

* [What is XGBoost?](https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/#h-what-is-xgboost)
* [Why Ensemble Learning?](https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/#h-why-ensemble-learning)
* [Demonstrating the Potential of Gradient Boosting](https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/#h-demonstrating-the-potential-of-gradient-boosting)
* [Using Gradient d=Descent for Optimizing the Loss Function](https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/#h-using-gradient-d-descent-for-optimizing-the-loss-function)
* [Unique Features of XGBoost](https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/#h-unique-features-of-xgboost)
* [Python Code for XGBoost](https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/#h-python-code-for-xgboost)
* [XGBoost Benefits and Attributes](https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/#h-xgboost-benefits-and-attributes)
* [End Notes](https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/#h-end-notes)

What is XGBoost?

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’s no wonder then that CERN recognized it as the best approach to classify signals from the Large Hadron Collider. This particular challenge posed by CERN required a solution that would be scalable to process data being generated at the rate of 3 petabytes per year and effectively distinguish an extremely rare signal from background noises in a complex physical process. XGBoost emerged as the most useful, straightforward and robust solution.

We recommend going through the below article as well to fully understand the various terms and concepts mentioned in this article:

* [A Comprehensive Guide to Ensemble Modeling (with Python codes)](https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/)
* [Ensemble Learning Course: Ensemble Learning and Ensemble Learning Techniques](https://courses.analyticsvidhya.com/courses/ensemble-learning-and-ensemble-learning-techniques?utm_source=blog&utm_medium=an-end-to-end-guide-to-understand-the-math-behind-xgboost)

Why Ensemble Learning?

XGBoost is an [ensemble learning](https://courses.analyticsvidhya.com/courses/ensemble-learning-and-ensemble-learning-techniques?utm_source=blog&utm_medium=an-end-to-end-guide-to-understand-the-math-behind-xgboost) method. Sometimes, it may not be sufficient to rely upon the results of just one machine learning model. Ensemble learning offers a systematic solution to combine the predictive power of multiple learners. The resultant is a single model which gives the aggregated output from several models.

The models that form the ensemble, also known as base learners, could be either from the same learning algorithm or different learning algorithms. Bagging and boosting are two widely used ensemble learners. Though these two techniques can be used with several statistical models, the most predominant usage has been with decision trees.

Let’s briefly discuss bagging before taking a more detailed look at the concept of gradient boosting.

1. Bagging

While decision trees are one of the most easily interpretable models, they exhibit highly variable behavior. Consider a single training dataset that we randomly split into two parts. Now, let’s use each part to train a decision tree in order to obtain two models.

When we fit both these models, they would yield different results. Decision trees are said to be associated with high variance due to this behavior. Bagging or boosting aggregation helps to reduce the variance in any learner. Several decision trees which are generated in parallel, form the base learners of bagging technique. Data sampled with replacement is fed to these learners for training. The final prediction is the averaged output from all the learners.

2. Boosting

In boosting, the trees are built sequentially such that each subsequent tree aims to reduce the errors of the previous tree. Each tree learns from its predecessors and updates the residual errors. Hence, the tree that grows next in the sequence will learn from an updated version of the residuals.

The base learners in boosting are weak learners in which the bias is high, and the predictive power is just a tad better than random guessing. Each of these weak learners contributes some vital information for prediction, enabling the boosting technique to produce a strong learner by effectively combining these weak learners. The final strong learner brings down both the bias and the variance.

**In contrast to bagging techniques like Random Forest, in which trees are grown to their maximum extent, boosting makes use of trees with fewer splits**. Such small trees, which are not very deep, are highly interpretable. Parameters like the number of trees or iterations, the rate at which the gradient boosting learns, and the depth of the tree, could be optimally selected through validation techniques like k-fold cross validation. **Having a large number of trees might lead to overfitting. So, it is necessary to carefully choose the stopping criteria for boosting.**

The gradient boosting [ensemble technique](https://courses.analyticsvidhya.com/courses/ensemble-learning-and-ensemble-learning-techniques?utm_source=blog&utm_medium=an-end-to-end-guide-to-understand-the-math-behind-xgboost) consists of three simple steps:

* An initial model F0 is defined to predict the target variable y. This model will be associated with a residual (y – F0)
* A new model h1 is fit to the residuals from the previous step
* Now, F0 and h1 are combined to give F1, the boosted version of F0. The mean squared error from F1 will be lower than that from F0:

https://cdn.analyticsvidhya.com/wp-content/uploads/2018/09/1.png

To improve the performance of F1, we could model after the residuals of F1 and create a new model F2:

https://cdn.analyticsvidhya.com/wp-content/uploads/2018/09/2.png

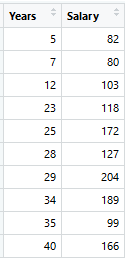
This can be done for *‘m’*iterations, until residuals have been minimized as much as possible:

https://cdn.analyticsvidhya.com/wp-content/uploads/2018/09/3.png

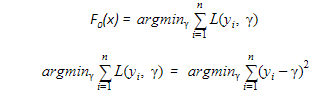
Here, the additive learners do not disturb the functions created in the previous steps. Instead, they impart information of their own to bring down the errors.

Demonstrating the Potential of Gradient Boosting

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 an [ensemble model](https://courses.analyticsvidhya.com/courses/ensemble-learning-and-ensemble-learning-techniques?utm_source=blog&utm_medium=an-end-to-end-guide-to-understand-the-math-behind-xgboost) 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.



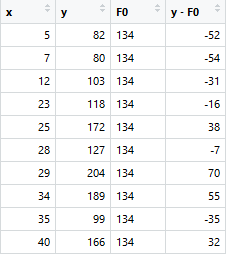
As the first step, the model should be initialized with a function F0(x). F0(x) should be a function which minimizes the loss function or MSE (mean squared error), in this case:



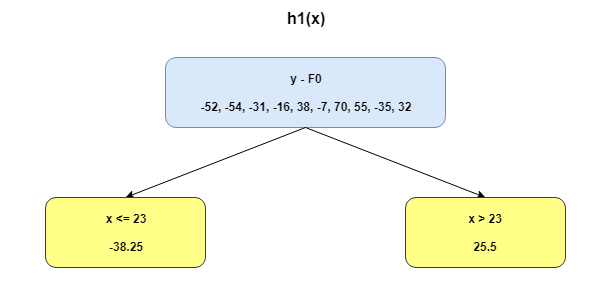
Taking the first differential of the above equation with respect to γ, it is seen that the function minimizes at the mean i=1nyin. So, the boosting model could be initiated with:

https://cdn.analyticsvidhya.com/wp-content/uploads/2018/09/6.png

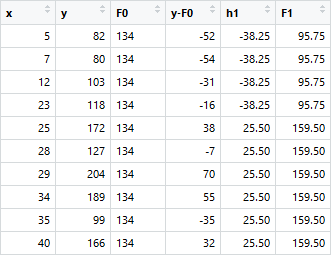
F0(x) gives the predictions from the first stage of our model. Now, the residual error for each instance is (yi – F0(x)).



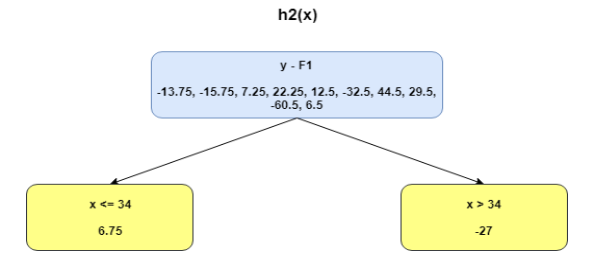
We can use the residuals from F0(x) to create h1(x). h1(x) will be a regression tree which will try and reduce the residuals from the previous step. The output of h1(x) won’t be a prediction of y; instead, it will help in predicting the successive function F1(x) which will bring down the residuals.

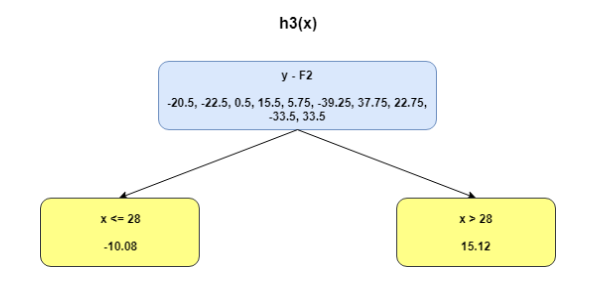


The additive model h1(x) computes the mean of the residuals (y – F0) at each leaf of the tree. The boosted function F1(x) is obtained by summing F0(x) and h1(x). This way h1(x) learns from the residuals of F0(x) and suppresses it in F1(x).



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).

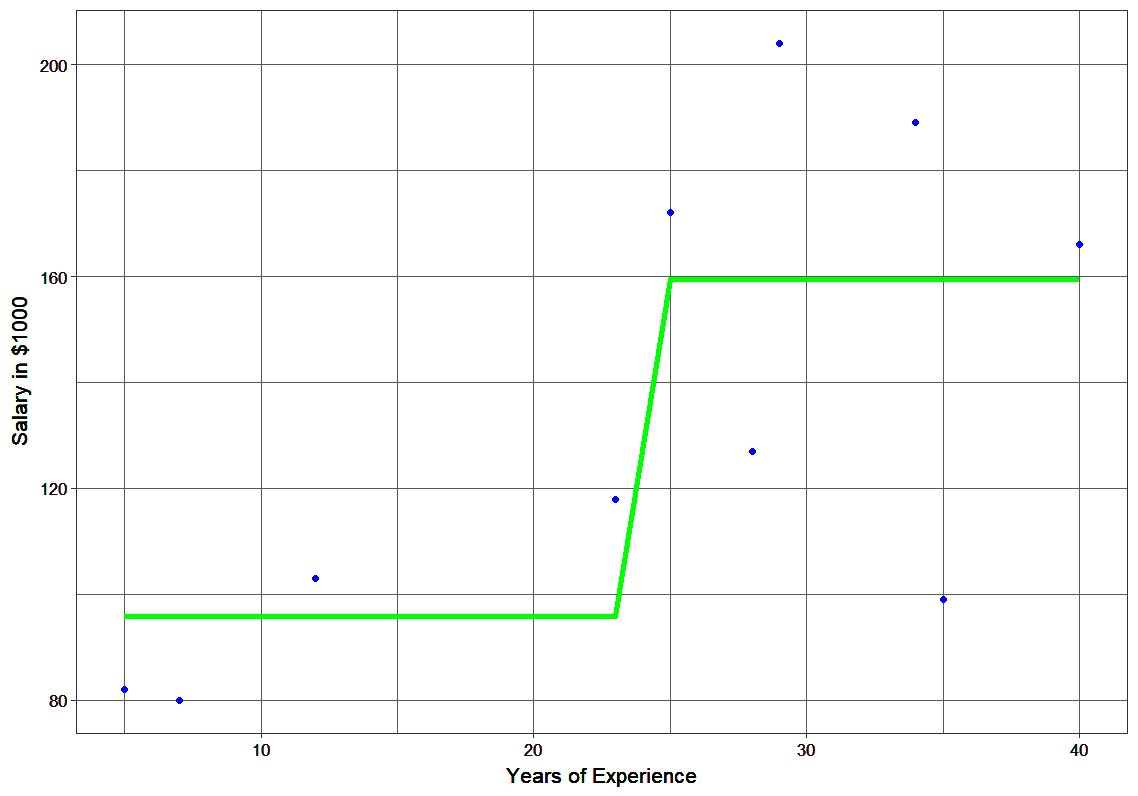


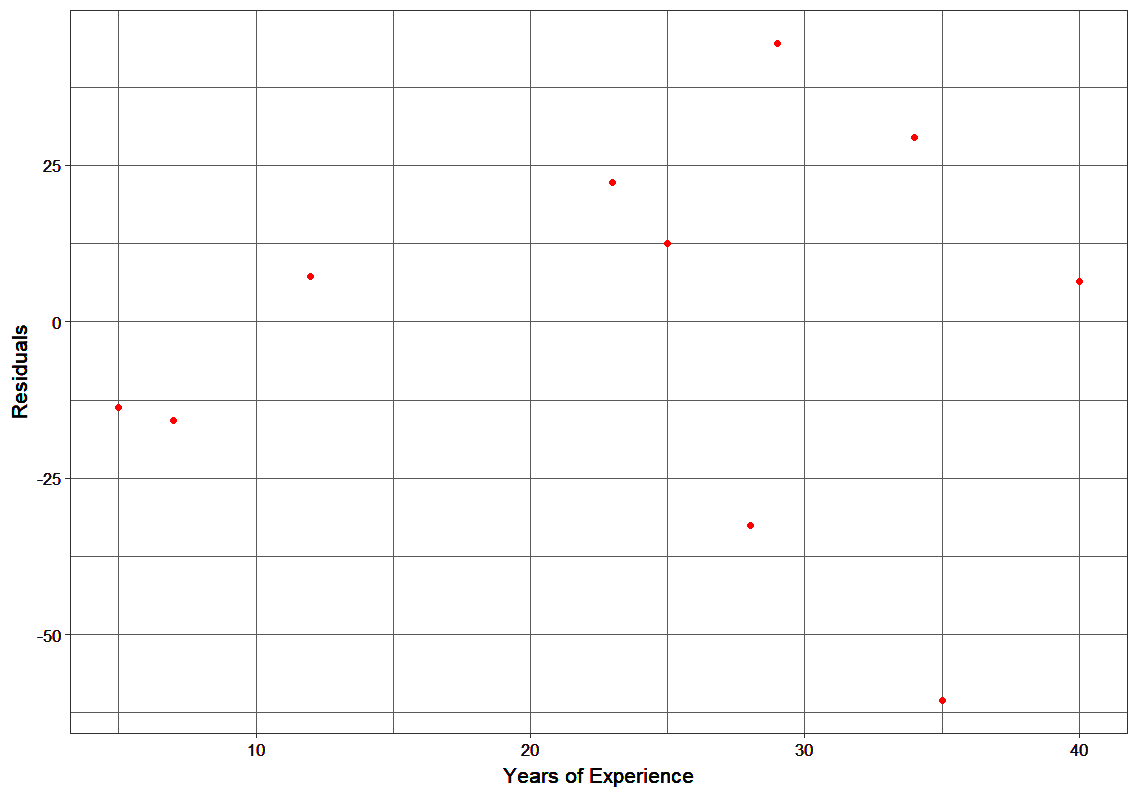


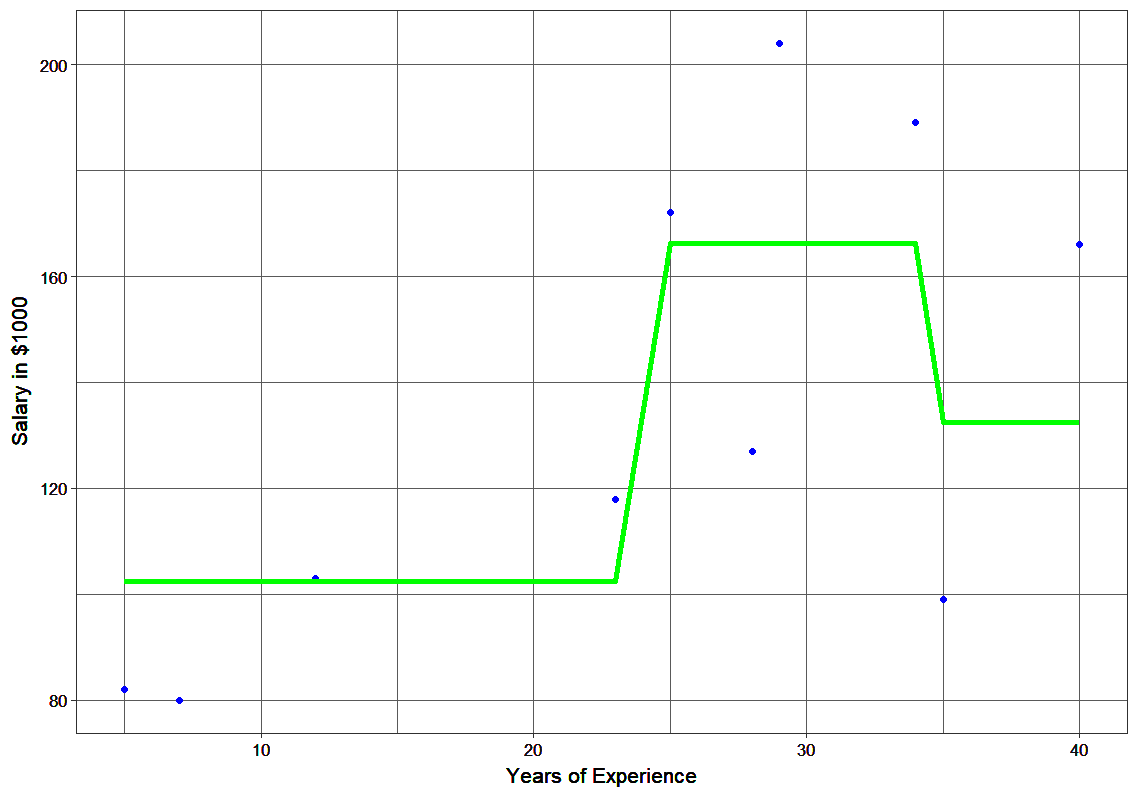


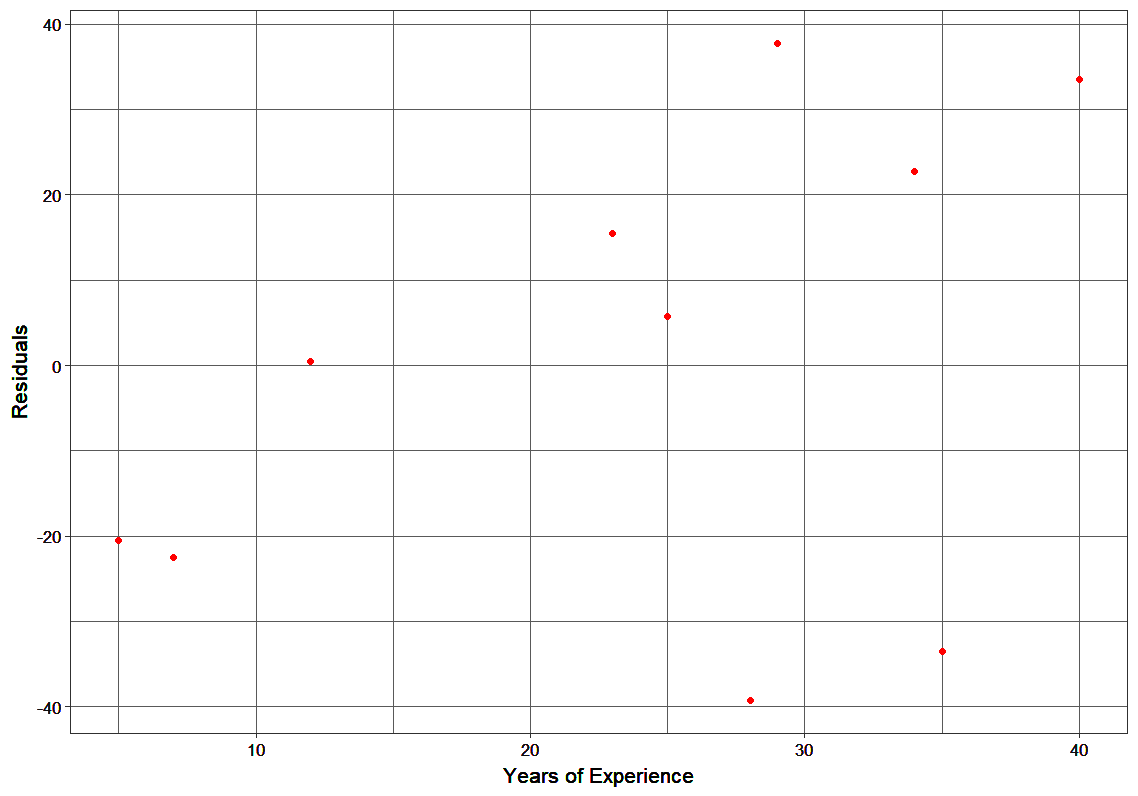
The MSEs for F0(x), F1(x) and F2(x) are 875, 692 and 540. It’s amazing how these simple weak learners can bring about a huge reduction in error!

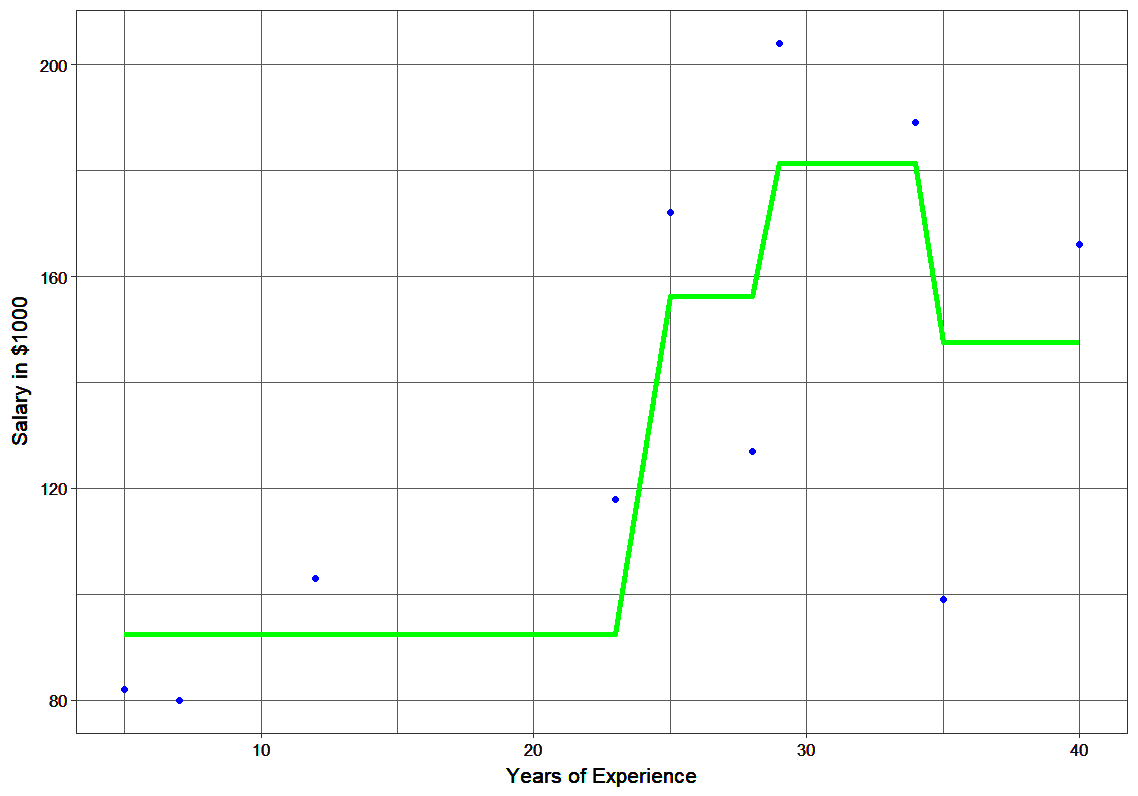
Note that each learner, hm(x), is trained on the residuals. All the additive learners in boosting are modeled after the residual errors at each step. Intuitively, it could be observed that the boosting learners make use of the patterns in residual errors. At the stage where maximum accuracy is reached by boosting, the residuals appear to be randomly distributed without any pattern.

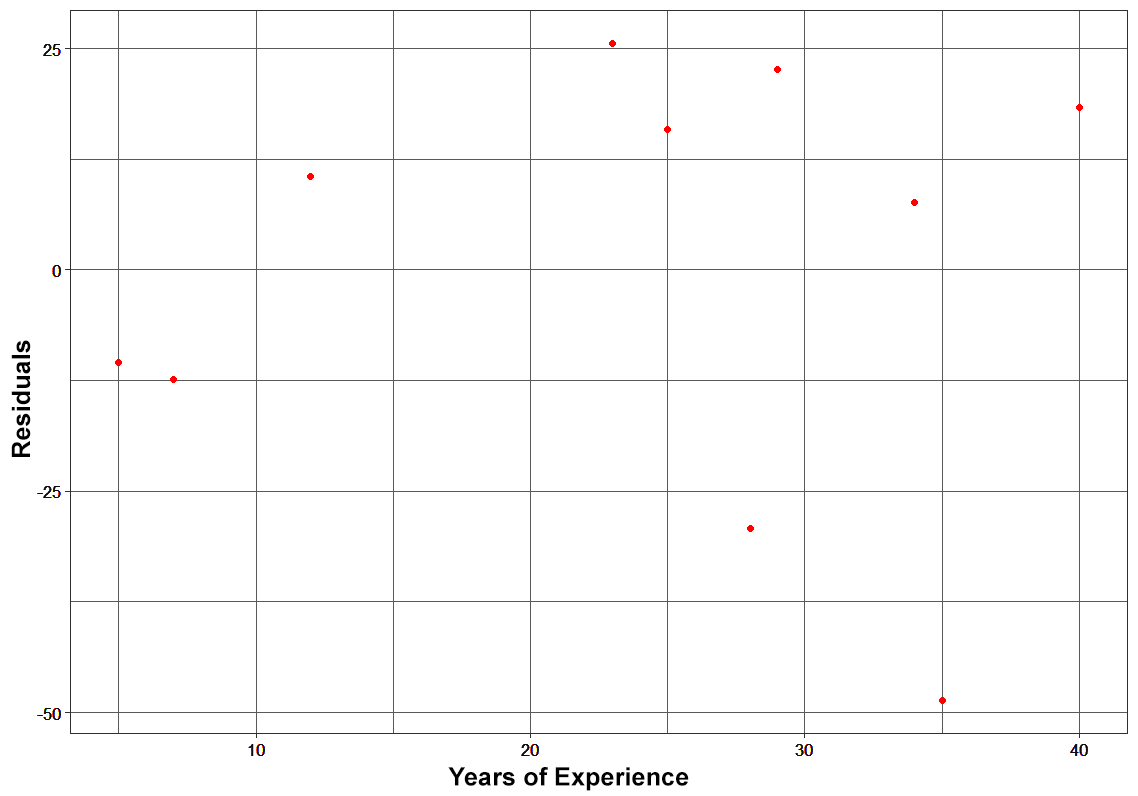












**Plots of Fn and hn**

Using Gradient d=Descent for Optimizing the Loss Function

In the case discussed above, MSE was the loss function. The mean minimized the error here. When MAE (mean absolute error) is the loss function, the median would be used as F0(x) to initialize the model. A unit change in y would cause a unit change in MAE as well.

For MSE, the change observed would be roughly exponential. Instead of fitting hm(x) on the residuals, fitting it on the gradient of loss function, or the step along which loss occurs, would make this process generic and applicable across all loss functions.

Gradient descent helps us minimize any differentiable function. Earlier, the regression tree for hm(x) predicted the mean residual at each terminal node of the tree. In gradient boosting, the average gradient component would be computed.

For each node, there is a factor γ with which hm(x) is multiplied. This accounts for the difference in impact of each branch of the split. Gradient boosting helps in predicting the optimal gradient for the additive model, unlike classical gradient descent techniques which reduce error in the output at each iteration.

The following steps are involved in gradient boosting:

* F0(x) – with which we initialize the boosting algorithm – is to be defined:

https://cdn.analyticsvidhya.com/wp-content/uploads/2018/09/17.png

* The gradient of the loss function is computed iteratively:

https://cdn.analyticsvidhya.com/wp-content/uploads/2018/09/18.png

* Each hm(x) is fit on the gradient obtained at each step
* The multiplicative factor γm for each terminal node is derived and the boosted model Fm(x) is defined:

https://cdn.analyticsvidhya.com/wp-content/uploads/2018/09/19.png

Unique Features of XGBoost

XGBoost is a popular implementation of gradient boosting. Let’s discuss some features of XGBoost that make it so interesting:

* **Regularization:**XGBoost has an option to penalize complex models through both L1 and L2 regularization. Regularization helps in preventing overfitting
* **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
* **Weighted quantile sketch:** Most existing tree based algorithms can find the split points when the data points are of equal weights (using quantile sketch algorithm). However, they are not equipped to handle weighted data. XGBoost has a distributed weighted quantile sketch algorithm to effectively handle weighted 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
* **Cache awareness:** In XGBoost, non-continuous memory access is required to get the gradient statistics by row index. Hence, XGBoost has been designed to make optimal use of hardware. This is done by allocating internal buffers in each thread, where the gradient statistics can be stored
* **Out-of-core computing:**This feature optimizes the available disk space and maximizes its usage when handling huge datasets that do not fit into memory

Python Code for XGBoost

Here’s a live coding window to see how XGBoost works and play around with the code without leaving this article!

XGBoost Benefits and Attributes

1. **High accuracy:** XGBoost is known for its accuracy and has been shown to outperform other machine learning algorithms in many predictive modeling tasks.
2. **Scalability:** XGBoost is highly scalable and can handle large datasets with millions of rows and columns.
3. **Efficiency:** XGBoost is designed to be computationally efficient and can quickly train models on large datasets.
4. **Flexibility:**XGBoost supports a variety of data types and objectives, including regression, classification, and ranking problems.
5. **Regularization:** XGBoost incorporates regularization techniques to avoid overfitting and improve generalization performance.
6. **Interpretability:**XGBoost provides feature importance scores that can help users understand which features are most important for making predictions.
7. **Open-source:** XGBoost is an open-source library that is widely used and supported by the data science community.

End Notes

So that was all about the mathematics that power the popular XGBoost algorithm. If your basics are solid, this article must have been a breeze for you. It’s such a powerful algorithm and while there are other techniques that have spawned from it (like CATBoost), XGBoost remains a game changer in the machine learning community. We highly recommend you to take up this course to sharpen your skills in machine learning and learn all the state-of-the-art techniques used in the field with our [Applied Machine Learning – Beginner to Professional](https://courses.analyticsvidhya.com/courses/applied-machine-learning-beginner-to-professional?utm_source=blog&utm_medium=content) course.

Frequently Asked Questions

**Q1. What is the XGBoost algorithm?**

A. XGBoost is a machine learning algorithm that uses an ensemble of decision trees and gradient boosting to make predictions. It is widely used in data science and has won several machine learning competitions.

**Q2. Is XGBoost better than random forest?**

A. The performance of XGBoost and random forest depends on the data and problem being solved. XGBoost tends to perform better on structured data, while random forest can be more effective on unstructured data.

**Q3. What is XGBoost Python used for?**

A. XGBoost Python is a Python package that enables building and training models using the XGBoost algorithm in Python. It includes many functions for tuning and optimizing model performance.

**Q4. Is XGBoost a classifier or regression?**

A. XGBoost is a versatile algorithm that can be used for both classification and regression problems. It can handle different types of data and can be customized for specific needs.