Member-only story

Log Book — XGBoost, the math behind the algorithm

This post deals with the math behind the XGBoost Algorithm in a, hopefully, easy way

dearC · Follow

Published in Towards Data Science · 8 min read · Apr 21, 2020

Source

> If you want to understand something well, try to explain it simply. — Feynman

XGBoost is a beautiful algorithm and the journey through it has been nothing short of illuminating. The concepts, often simple and beautiful gets lost in mathematical jargon. I had faced the same challenges while understanding the math and this is an attempt to consolidate my understanding while helping others on a similar journey.

In order to understand what XGBoost does we will need to understand what Gradient Boosting is, so let's first understand the concepts behind that. Please note that this post assumes familiarity with the boosting process in general and will just try to touch upon the intuition and math behind Gradient Boosting and XGBoost. Let's jump right in.

Understanding Gradient Boosting

Step 1 — the initial function

As always let's start with a crude initial function F_0, something like average of all values in case of regression. It will give us some output, however bad.

Step 2 — the loss function

Next we will compute the Loss function given by $L(y_i, F_t(x_i))$.

Now what is a loss function? It is nothing but a way to measure **the difference between actual and predicted values.** Few examples would be:

Square loss

$$L(y, F(x)) = (y - F(x))^2$$

Absolute loss (more robust to outliers)

$$L(y, F) = |y - F|$$

Huber loss (more robust to outliers)

$$L(y, F) = \begin{cases} \frac{1}{2}(y - F)^2 & |y - F| \leq \delta \\ \delta(|y - F| - \delta/2) & |y - F| > \delta \end{cases}$$

Source

And why this robustness to outliers is important can be understood from the below table:

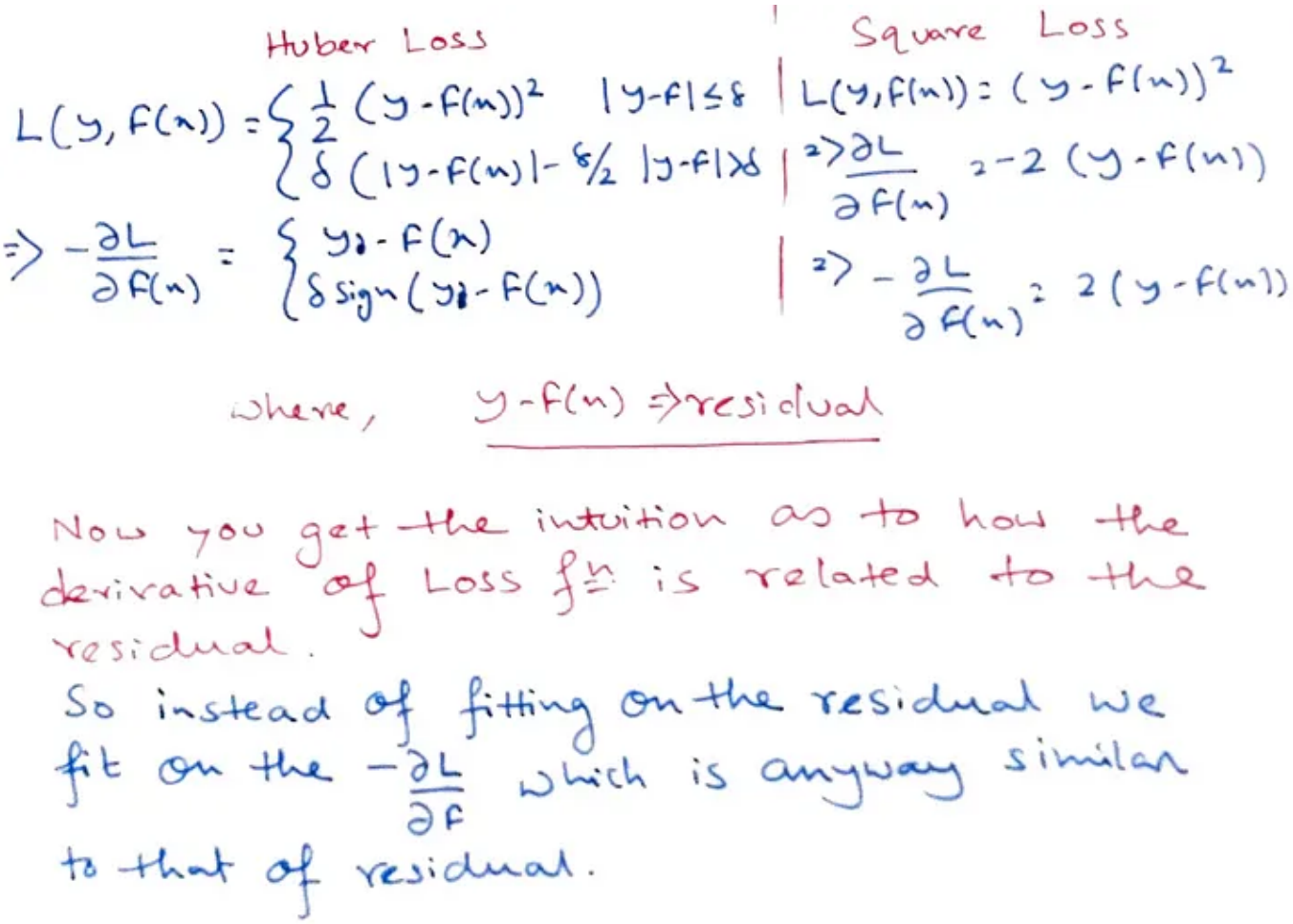
y_i	0.5	1.2	2	5*
$F(x_i)$	0.6	1.4	1.5	1.7
Square loss	0.005	0.02	0.125	5.445
Absolute loss	0.1	0.2	0.5	3.3
Huber loss($\delta = 0.5$)	0.005	0.02	0.125	1.525

Outliers and its impact on Loss Function, here 5 is the outlier. Check the values of different Loss functions

The idea is that lower the value of the Loss Function the more accurate our predictions are, so now getting better predictions has become a minimization problem of the Loss function.

Step 2 — the new targets

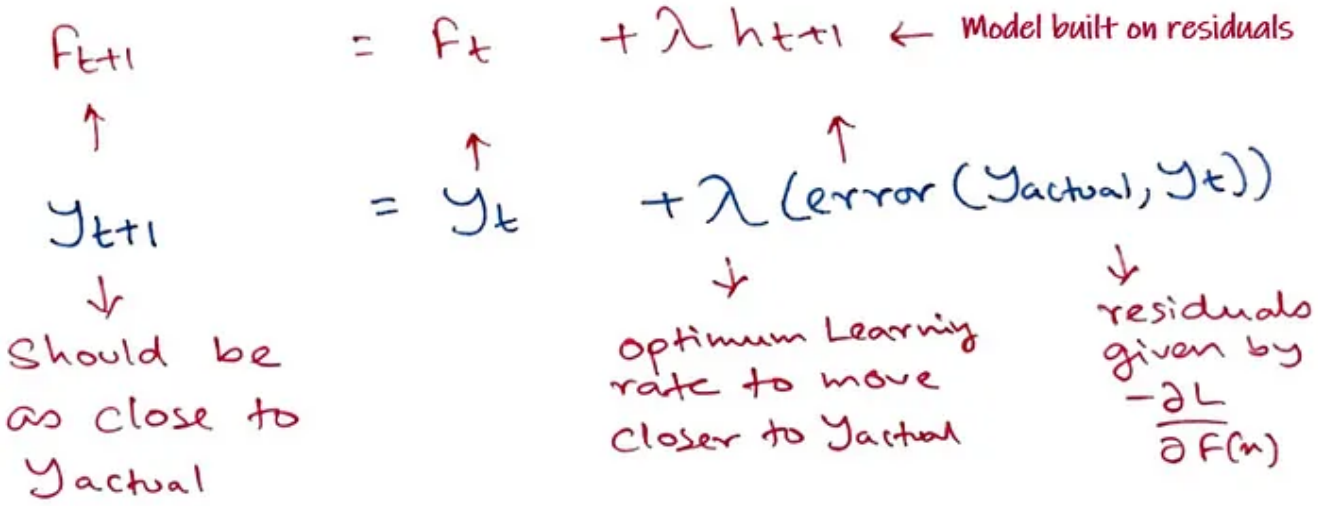
So far we have the built our initial model, taken its predictions. **Next, we should have fit a new model on the residuals given by the Loss function, but there is a subtle twist: we will instead fit on the negative gradient of the loss function**, the intuition of why we are doing that and why they are similar is given below:



The gradient can be interpreted as the "direction and rate of fastest increase" of a function, so the negative gradient tells us the direction of the minima of a function, in this case which is the minima of the loss function.

We will follow the same approach of gradient descent here. Take steps towards the minima of the Loss function, and **how big or small a step will be given by the learning rate of the algorithm. And at the minima of the loss function we will have the lowest error rate.**

So we will build a new model h_{t+1} on the -ve gradient of the loss function.



Step 3— the additive approach

This process of fitting the model iteratively on the -ve gradient will continue till we have **reached the minima or the limit of the number of weak learners given by T**, this is called the additive approach

$$f_0(x_i) = 0$$
$$f_1(x_i) = f_0(x_i) + h_1(x_i)$$
$$f_2(x_i) = f_0(x_i) + h_1(x_i) + h_2(x_i)$$
$$\vdots$$
$$f_t(x_i) = f_0(x_i) + \sum_{t=1}^{T} h_t(x_i) \Rightarrow f_{t-1}(x_i) + h_t(x_i)$$

Additive Approach

Recall that, in Adaboost,"shortcomings" are identified by high-weight data points. In Gradient Boosting, "shortcomings" are identified by gradients.

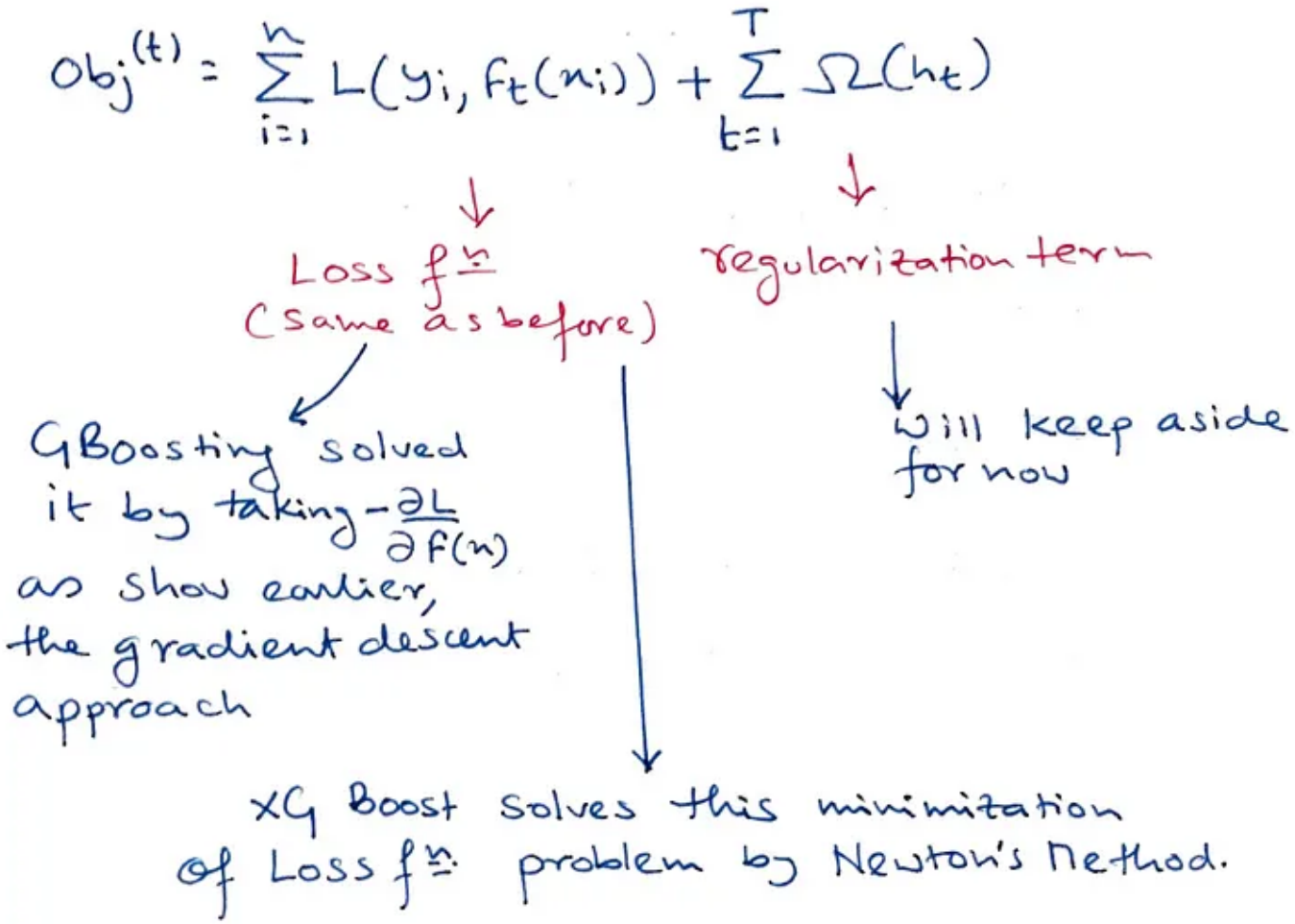
This is in short of the intuition as to how Gradient Boosting works. **In case of regression and classification the only thing that differs is the loss function**

that is used.

Next we will see how XGBoost differs from Gradient Boosting.

XGBoost

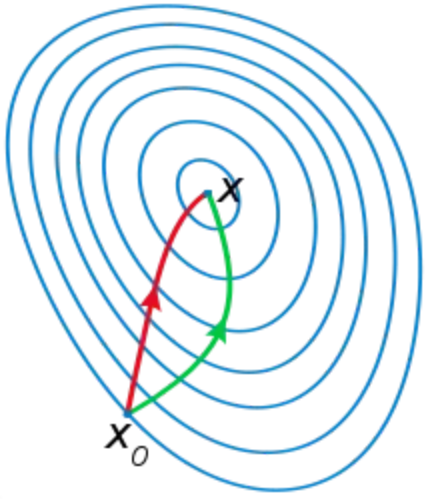
XGBoost and GBM both follow the principle of gradient boosted trees, but **XGBoost uses a more regularized model formulation to control over-fitting, which gives it better performance**, which is why it's also known as 'regularized boosting' technique.



Newton's Method

So what is this Newton's method? **In Stochastic Gradient Descent, we use less point to take less time to compute the direction we should go towards, in order to make more of them, in the hope we go there quicker. In Newton's method, we take more time to compute the direction we want to go into, in the hope we have to take fewer steps in order to get there.**

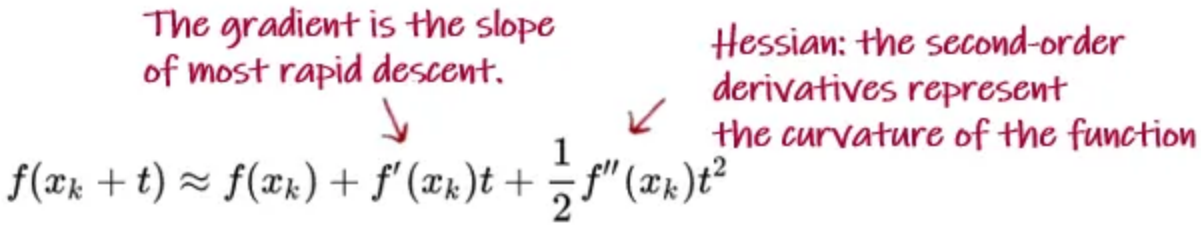
> One important point to note is that, even in the case of Gradient Boosting while the regression problem was solved using gradient descent, the classification problem still uses Newton's method to solve the minimization problem. XGBoost uses this method in both the cases of classification and regression.



A comparison of gradient descent (green) and Newton's method (red) for minimizing a function (with small step sizes). Newton's method uses curvature information (i.e. the second derivative) to take a more direct route. (source)

Newton's method attempts to solve the minimization problem by constructing a sequence $\{x_k\}$ from an initial guess (starting point) $x_0 \in R$ that

converges towards a minimizer x* of *f* by using a sequence of second-order Taylor approximations of *f* around the iterates. The second-order Taylor expansion of *f* around $\{x_k\}$ is



(source)

The 2nd order derivatives are important in speeding gradient descent because if your algorithm is leading you zig-zag across a valley so that you are making very little progress down the actual slope of the valley, instead just step wise ending up repeatedly crossing this valley, adjusting the direction by the 2nd order derivative will skew your direction of descent in the direction of this valley, thus converting a slow descent into a much more rapid one.

Loss Function

We had seen how the Square Loss behaves in the gradient boosting framework, let's take a quick look what becomes of the square loss function in the XGBoost approach:

$$obj^{t} = \sum_{i=1}^{n} L(y_i, F_t(x_i)) + \sum_{t=1}^{T} \Omega(h_t)$$

$$= \sum_{i=1}^{n} L(y_i, F_{t-1}(x_i) + h_t(x_i)) + \sum_{t=1}^{T} \Omega(h_t)$$

Loss $f^{n} \Rightarrow$ Square Loss

$$= \sum_{i=1}^{n} (y_i - (f_{t-1}(x_i) + h_t(x_i)))^2 + \sum_{t=1}^{T} \Omega(h_t)$$

Comparing with the Newton's formula →

$t = h_t(x_i)$

$x = f_{t-1}(x_i)$

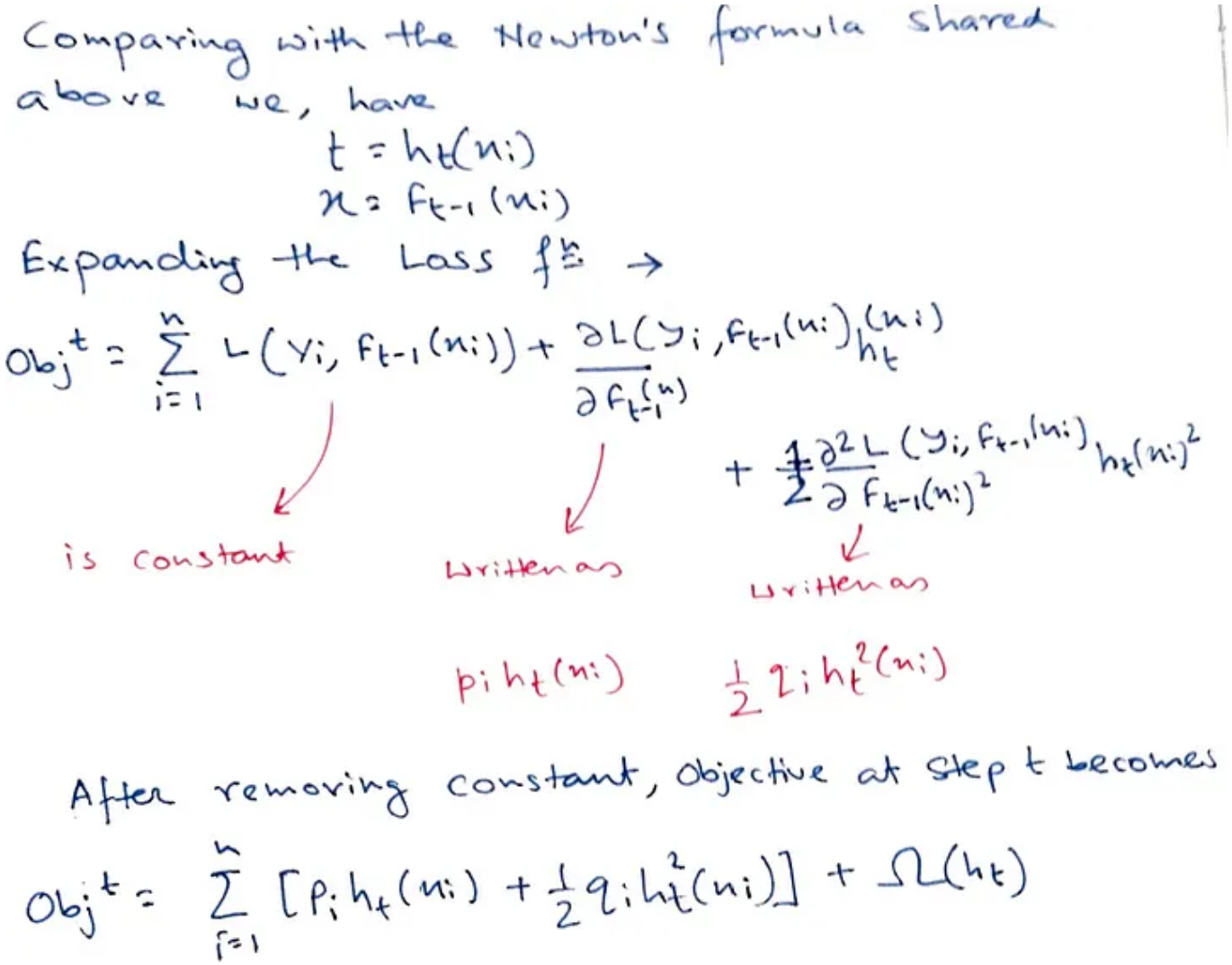
$$= \sum_{i=1}^{n} \left[(y_i - F_{t-1}(x_i))^2 + 2(f_{t-1}(x_i) - y_i) h_t(x_i) + h_t(x_i)^2 \right] + \sum_{t=1}^{T} \Omega(h_t)$$

$f(x_k)$ $f'(x_k)t$ $\frac{1}{2} f''(x_k)t^2$

This is constant residual^2

if you compare it with gradient method this is the one used there $\Rightarrow -\frac{\partial L}{\partial F(x)}$ -ve gradient

> *The form of MSE is friendly, with a first order term (usually called the residual) and a quadratic term. For other losses of interest (for example, logistic loss), it is not so easy to get such a nice form. So in the general case, we take the Taylor expansion of the loss function up to the second order — XGBoost Docs*

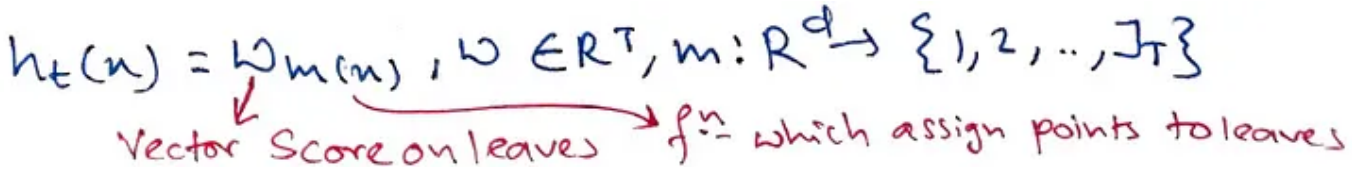


> *This becomes our optimization goal for the new tree. One important advantage of this definition is that the value of the objective function only depends on p_i and q_i. This is how XGBoost supports custom loss functions. We can optimize every loss function, including logistic regression and pairwise ranking, using exactly the same solver that takes p_i and q_i as input! — XGBoost Docs*

Regularization

Next we will deal with the Regularization term, but before going there we need to understand how a decision tree is defined mathematically. Intuitively if you think a decision tree is or mainly its output is a combination

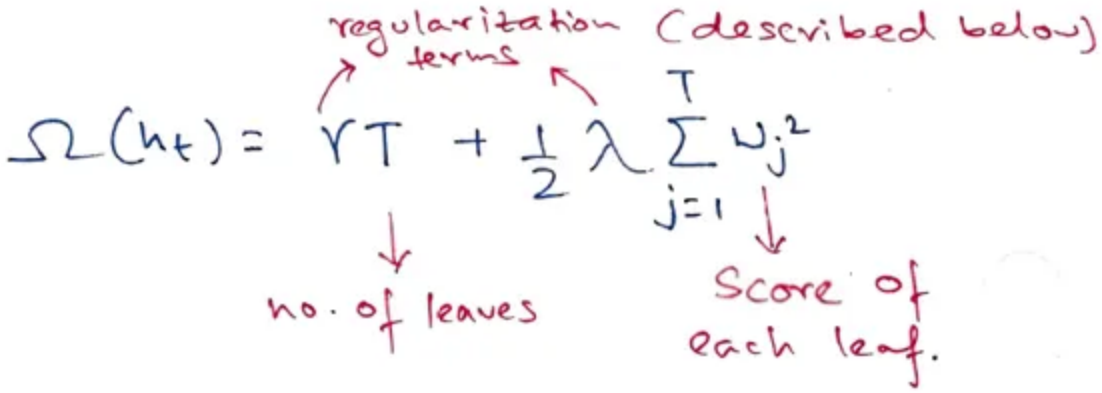
of leaves and a function assigning the data point to those leaves. Mathematically it is written as:



where JT is the number of leaves. This definition describes the prediction process on a tree as:

1. *Assign the data point to x to a leaf by m*
2. *Assign the corresponding score $w_{m(x)}$ on the m(x)-th leaf to the data point.*

In XGBoost, complexity is defined as:



These hyper-parameters in XGBoost are described as below:

`gamma` [default=0, alias: `min_split_loss`]

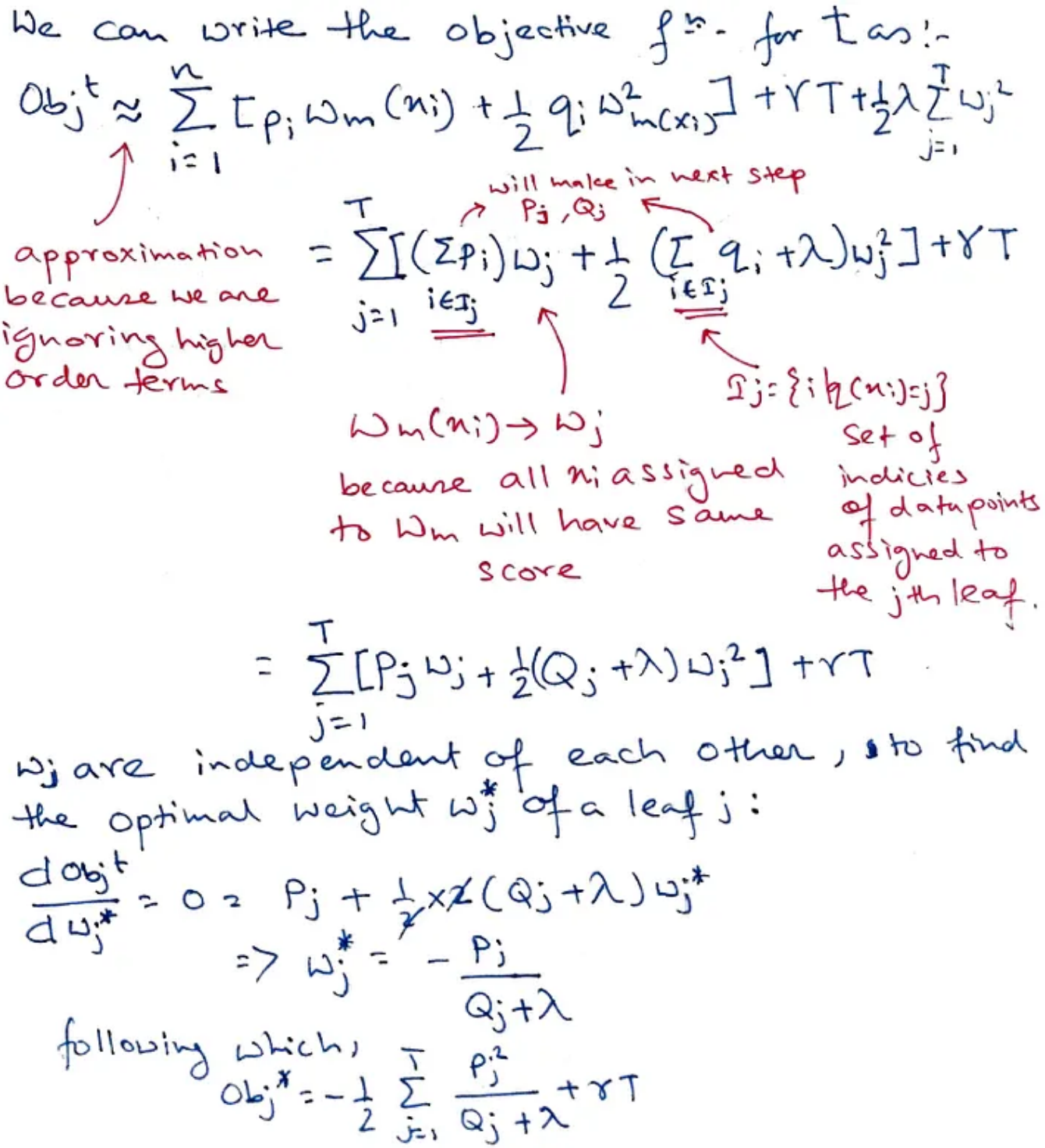
- Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger `gamma` is, the more conservative the algorithm will be.
- range: [0,∞]

`lambda` [default=0, alias: `reg_lambda`]

- L2 regularization term on weights. Increasing this value will make model more conservative. Normalised to number of training examples.

XGBoost Regularization Parameters

> *Of course, there is more than one way to define the complexity, but this one works well in practice. The regularization is one part most tree packages treat less carefully, or simply ignore. This was because the traditional treatment of tree learning only emphasized improving impurity, while the complexity control was left to heuristics. By defining it formally, we can get a better idea of what we are learning and obtain models that perform well in the wild. — XGBoost Docs*



$$Obj^t \approx \sum_{i=1}^{n} [p_i w_m(x_i) + \frac{1}{2} q_i w^2_{m(x_i)}] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

$$= \sum_{j=1}^{T} [(\sum_{i \in I_j} p_i) w_j + \frac{1}{2} (\sum_{i \in I_j} q_i + \lambda) w_j^2] + \gamma T$$

$$= \sum_{j=1}^{T} [P_j w_j + \frac{1}{2}(Q_j + \lambda) w_j^2] + \gamma T$$

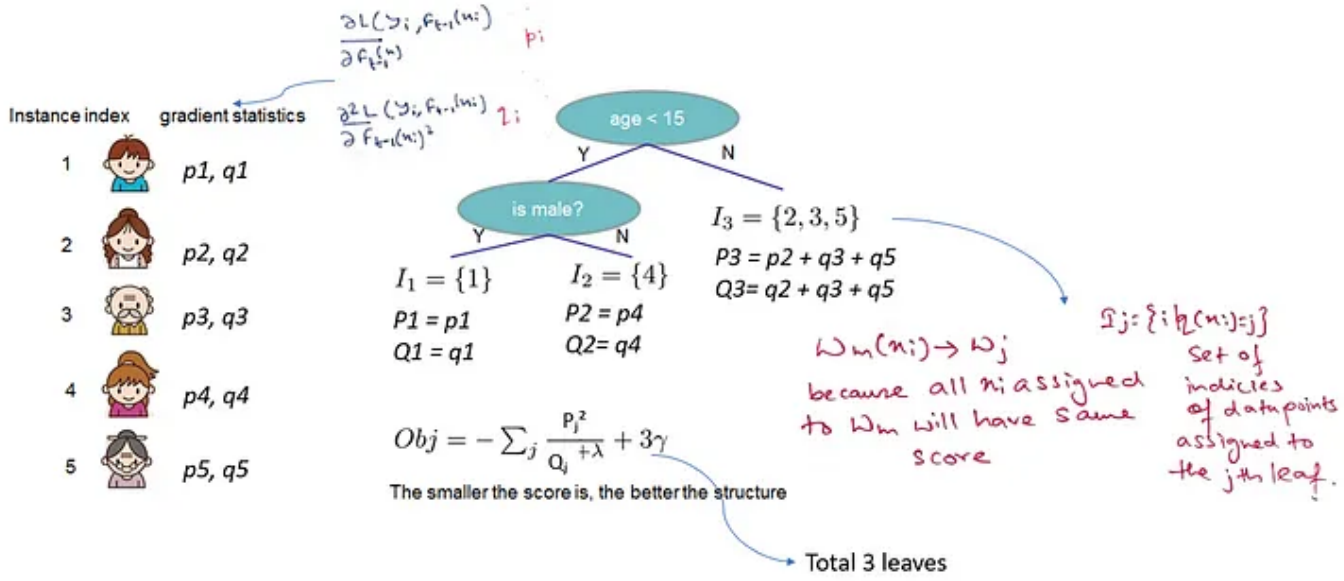
$$\frac{dObj^t}{dw_j^*} = 0 = P_j + \frac{1}{2} \times 2 (Q_j + \lambda) w_j^*$$

$$\Rightarrow w_j^* = -\frac{P_j}{Q_j + \lambda}$$

$$Obj^* = -\frac{1}{2} \sum_{j=1}^{T} \frac{P_j^2}{Q_j + \lambda} + \gamma T$$

The last equation measures *how good* a tree structure is.

If all this sounds a bit complicated, let's take a look at the picture, and see how the scores can be calculated.

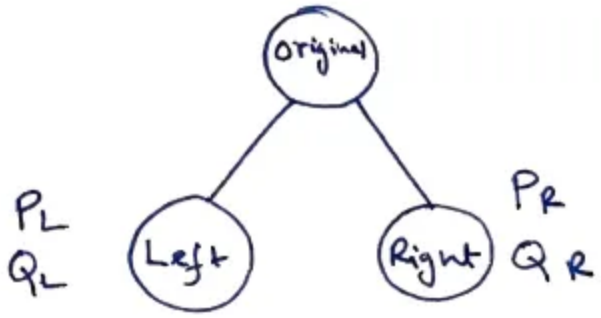


source

> *Basically, for a given tree structure, we push the statistics* gi *and* hi *to the leaves they belong to, sum the statistics together, and use the formula to calculate how good the tree is.* ***This score is like the impurity measure in a decision tree, except that it also takes the model complexity into account.*** *— XGBoost Docs*

Learn the tree structure

Now that we have a way **to measure how good a tree is, ideally we would enumerate all possible trees and pick the best one. In practice this is intractable, so we will try to optimize one level of the tree at a time.** Specifically we try to split a leaf into two leaves, and the score it gains is



$$Gain = \frac{1}{2}\left[\frac{P_L^2}{Q_L+\lambda} + \frac{P_R^2}{Q_R+\lambda} - \frac{(P_L+P_R)^2}{Q_L+Q_R+\lambda}\right] - \gamma$$

> *This formula can be decomposed as 1) the score on the new left leaf 2) the score on the new right leaf 3) The score on the original leaf 4) regularization on the additional leaf.* ***We can see an important fact here: if the gain is smaller than γ, we would do better not to add that branch.*** *This is exactly the* ***pruning*** *techniques in tree based models! By using the principles of supervised learning, we can naturally come up with the reason these techniques work. — XGBoost Docs*

Great now hopefully we have a preliminary understanding of what XGBoost and why it works the way it does. See you in the next post!

References

https://xgboost.readthedocs.io/en/latest/tutorials/model.html
https://drive.google.com/file/d/1CmNhi-7pZFnCEOJ9g7LQXwuIwom0ZN_D/view

https://arxiv.org/pdf/1603.02754.pdf
http://www.ccs.neu.edu/home/vip/teach/MLcourse/4_boosting/slides/gradient_boosting.pdf
https://mlexplained.com/2018/02/02/an-introduction-to-second-order-optimization-for-deep-learning-practitioners-basic-math-for-deep-learning-part-1/
http://learningsys.org/papers/LearningSys_2015_paper_32.pdf
https://www.youtube.com/user/joshstarmer

Data Science | Mathematics | Machine Learning

Written by dearC

Follow

158 Followers · Writer for Towards Data Science

Data Science @ Dell, writer @ https://book.thedatascienceinterviewproject.com/

More from dearC and Towards Data Science

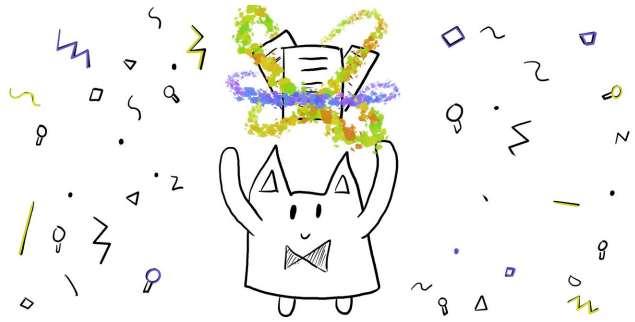
dearC in Towards Data Science

Log Book—Guide to Distance Measuring Approaches for K-...

In this guide I have tried to cover the different types and features of distances that can be...

· 9 min read · Jul 14, 2019

164 3



Adrian H. Raudaschl in Towards Data Science

Forget RAG, the Future is RAG-Fusion

The Next Frontier of Search: Retrieval Augmented Generation meets Reciprocal...

· 10 min read · Oct 6

830 14

Antonis Makropoulos in Towards Data Science

How to Build a Multi-GPU System for Deep Learning in 2023

This story provides a guide on how to build a multi-GPU system for deep learning and...

10 min read · Sep 17

553 11

dearC

A/B Testing—an summary

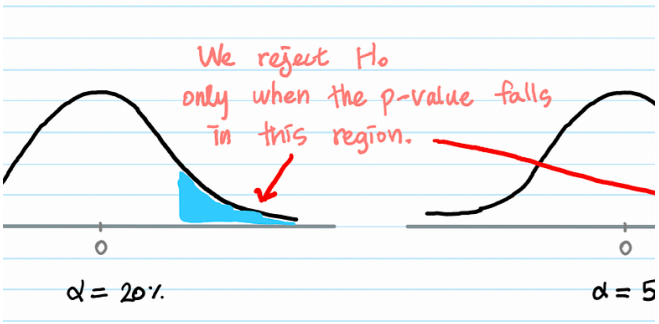
Recently I started reading up on A/B testing in preparation for a product-based Data Scienc...

· 11 min read · 4 days ago

9

See all from dearC See all from Towards Data Science

Recommended from Medium



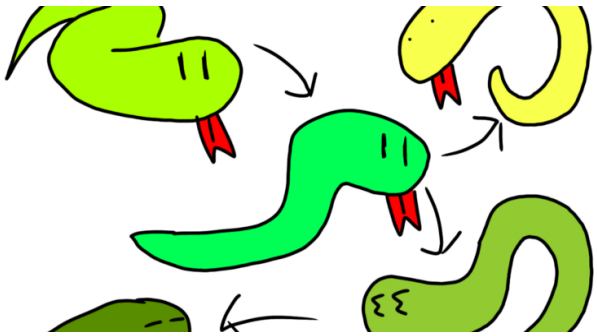
Ms Aerin in IntuitionMath

Chi Square Test—Intuition, Examples, and Step-by-Step...

The best way to see if two variables are related.

· 15 min read · Feb 13

408 3



Liu Zuo Lin in Level Up Coding

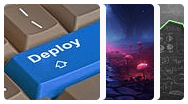
20 Python Concepts I Wish I Knew Way Earlier

Stuff I wish I learnt earlier as a beginner

· 9 min read · Apr 16

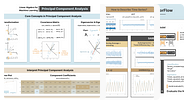
10.7K 35

Lists



Predictive Modeling w/ Python

20 stories · 482 saves



Practical Guides to Machine Learning

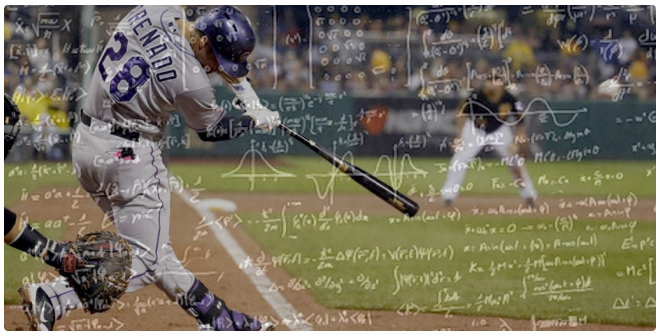
10 stories · 554 saves

Natural Language Processing

698 stories · 309 saves

New_Reading_List

174 stories · 147 saves



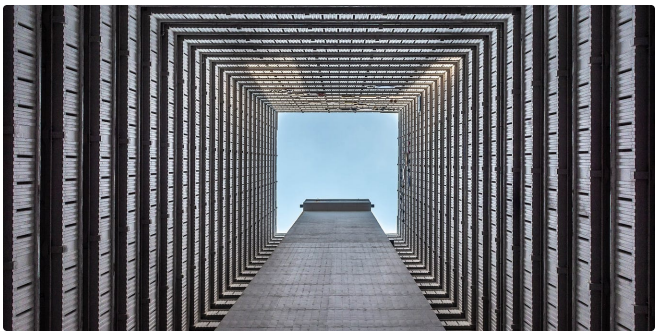
Sam Wirth in Hoyalytics

XGBoost: Theory and Application

Introduction

10 min read · Apr 18

11



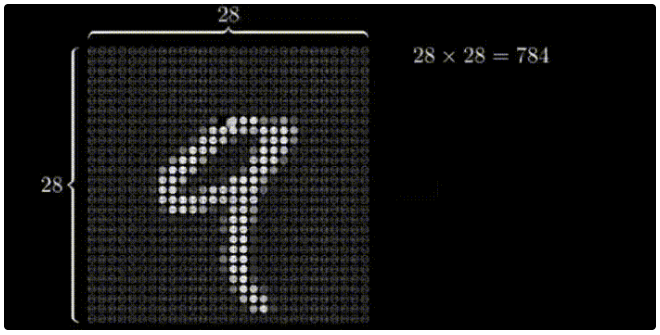
Rukshan Pramoditha in Data Science 365

Determining the Right Batch Size for a Neural Network to Get Bette...

Guidelines for choosing the right batch size to maintain optimal training speed and accurac...

4 min read · Sep 27, 2022

53



Sadaf Saleem

Neural Networks in 10mins. Simply Explained!

What are Neural Networks?

9 min read · May 15

Nick Wignall

4 Habits Confident People Avoid

#1: Asking for reassurance

9 min read · May 6

 508  2   8.8K  209 

See more recommendations