**XGBoost — How does this work**

[[Prathamesh Sonawane](https://medium.com/@prathameshsonawane?source=post_page-----e1cae7c5b6cb--------------------------------)](https://medium.com/@prathameshsonawane?source=post_page-----e1cae7c5b6cb--------------------------------)

[Prathamesh Sonawane](https://medium.com/@prathameshsonawane?source=post_page-----e1cae7c5b6cb--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2F41dc5a50d2ef&operation=register&redirect=https%3A%2F%2Fmedium.com%2F%40prathameshsonawane%2Fxgboost-how-does-this-work-e1cae7c5b6cb&user=Prathamesh+Sonawane&userId=41dc5a50d2ef&source=post_page-41dc5a50d2ef----e1cae7c5b6cb---------------------post_header-----------)

10 min read

·

Dec 5, 2023

9

I’ll cover everything there is to cover about XGBoost in this blog. Lmk if you think something is missing in the comments.

XGBoost (eXtreme Gradient Boosting) is an advanced implementation of gradient boosting algorithm. It’s a powerful machine learning algorithm especially popular for structured or tabular data. XGBoost has gained fame for its performance in a wide range of machine learning competitions and tasks.

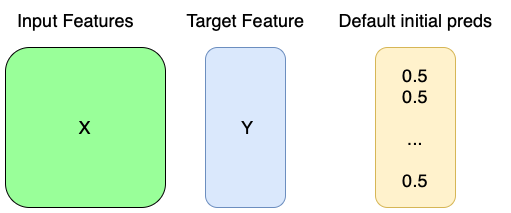
Here are some key aspects of XGBoost:

1. Gradient Boosting Framework:  
   It works by sequentially adding simple models to correct the errors made by previous models.
2. Efficiency and Scalability:  
   It is optimized to be highly computationally efficient and can handle large datasets.
3. Handling missing Data:  
   XGBoost has an in-built routine to handle missing data. (More on this later)
4. Tree pruning:  
   More on this later.
5. High Performance:  
   XGBoost is often the algorithm of choice for winning competitions.

Note — XgBoost is used for both Regression and Classification.

**Regression**

Let’s cover regression first then we can use a lot of it’s content to explain classification.



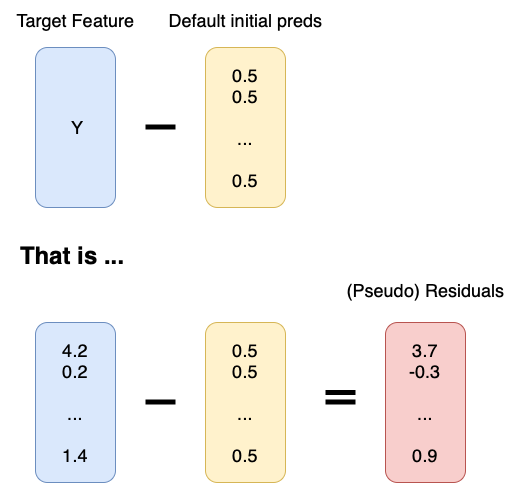
Initial data and default preds that we start with.

**I. Given data and initial predictions**

We are given input features (X) and target feature (Y). Now we start with a default set of predictions (by default set to 0.5 in both classification and regression but you can start from other values as well)

**II. Calculating Pseudo Residuals**

Calculate the (pseudo) residuals by subtracting Y from default initial preds.



Calculating (pseudo) residuals

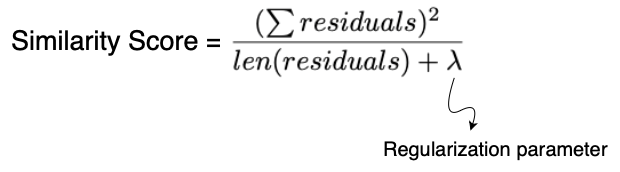
**III. Building XGBoost Trees (main part of algo.)**

1. **Start with all residuals in the same leaf.**



Ex. of residuals all in same node at the start.

**2. Calculate the similarity score using the following formula.**

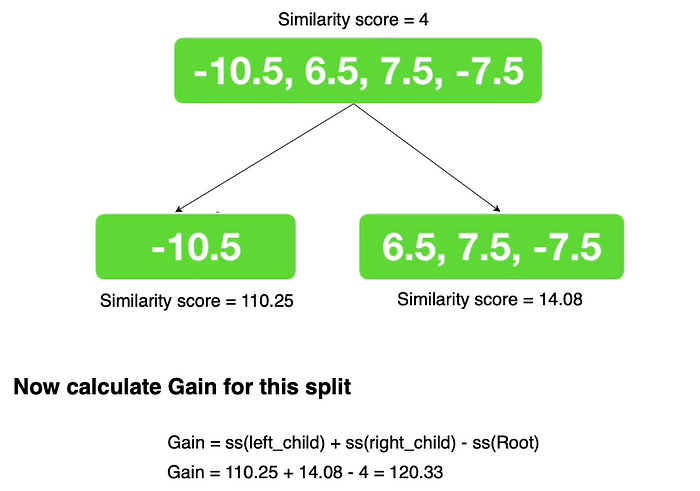


Formula to calculate similarity score

**3. Finding the best splitting feature and its values.**- Now we go through all the input features one by one.  
- Select the first feature, sort it’s values in ascending order and go through those values one by one.  
- Take 2 points at a time from the start, get the mean of those 2 values and then divide the leaf residuals according to the mean value, i.e. put residuals of elements with less than mean feature value to one node and the others to a different node.  
- Do this for all features and all values in each feature.

*Note — If the dataset is huge we can see that this step will end up taking a lot of time. To tackle that we can split using weighted quantiles (more on this later)*

**4. Calculating Gain**We now calculate the gain value for each of the splits created in step 3 as shown below.

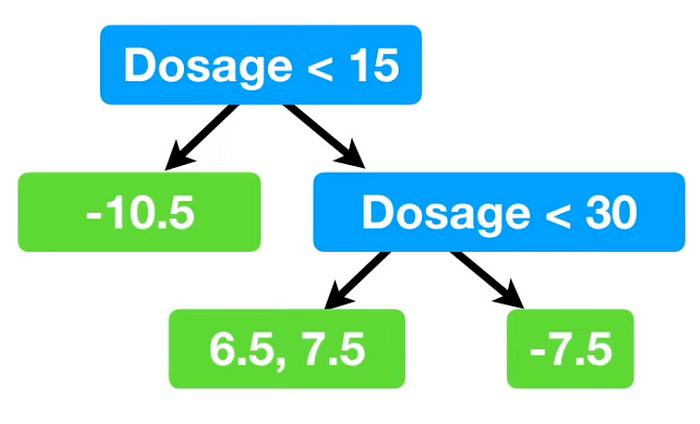


Calculating the gain for a split.

We go through all of the splits in step 3 and then take the split which gave us the highest gain. i.e. we select the one which best splits the observations.

**5. Creating the tree**We continue doing step 3 & 4 for the children of the tree to split it further. The stopping conditions are either the max depth has been reached (6 by default) or the leaf has just minimum amount of residuals in it (default = 1. Cover is used to determine this value. Discussed in later sections).

So after some time we may get a tree like this

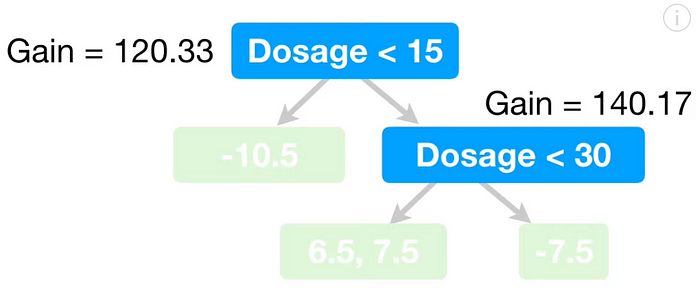


A XGB tree with max depth set to 2 (for illustration purposes)

**6. Pruning the tree**

We use a hyper parameter gamma(r) to prune trees.

We go bottom up from the tree while pruning. If the Gain of a parent node is less than the gamma (r) value then we prune its children (more formally if gain-r < 0). We only go up if we pruned at that particular point else we stop there.

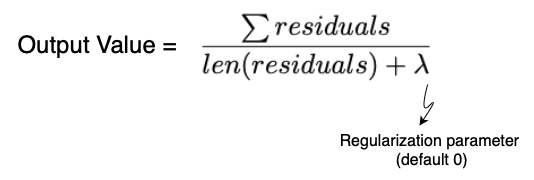


In this case of gamma was 130 we would not prune (dosage < 30) since gamma is less than the gain value. Since we didnt prune it we cant go to its parent i.e. (dosage < 15) to check for pruning.

*Note — setting gamma to 0 doesnt turn off pruning because there may be nodes with negative gain values, and gain-r will be < 0 in that case, leading to pruning.*

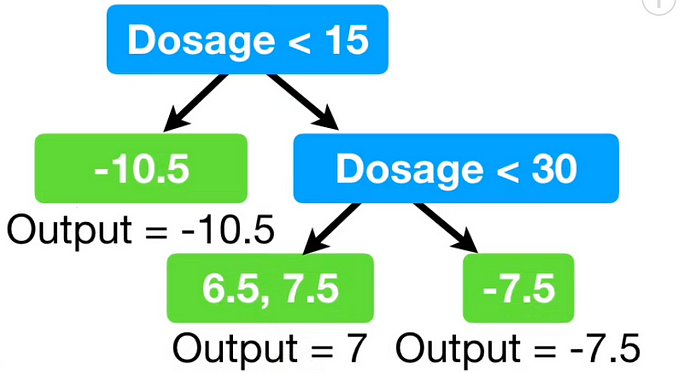
**7. Calculating the output Value**

Now we’ll calculate the output values for each of the child nodes in the tree using the following formula.



Formula to calculate the output value (regression)

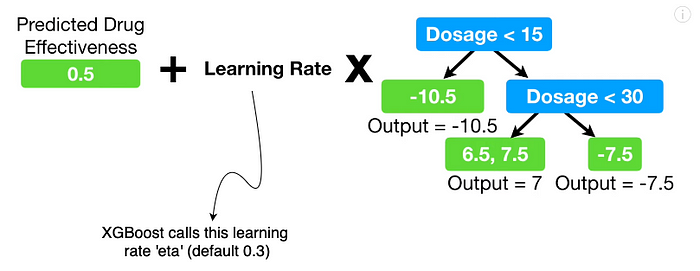
Using this calculate the output values for each child node. Illustrated below.



*Finally the first tree is complete!!!*

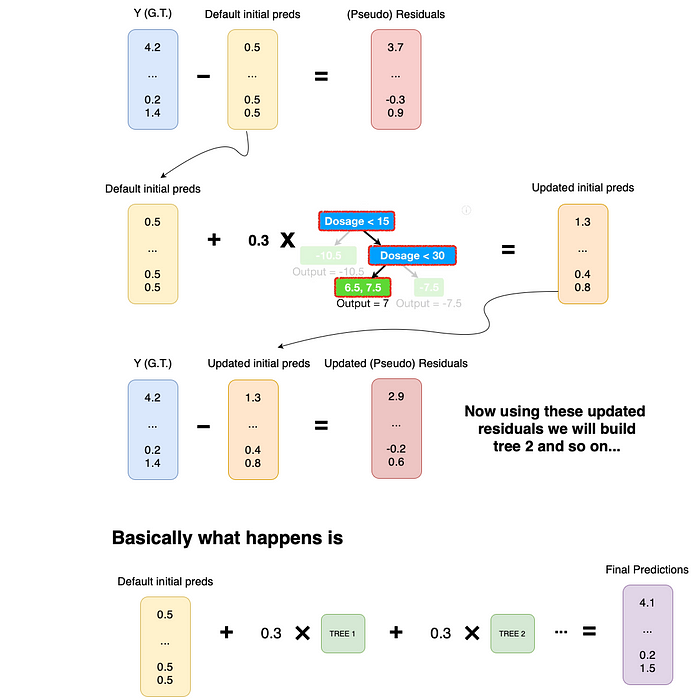
**IV. Updating the residuals and getting the output**

Now that we have a tree that can predict residuals, we can update are initial default vector of 0.5 predictions by adding these residuals to it (multiplied by a learning rate ofcourse. We don’t want to directly go to the pred value and overfit)



How the initial pred are updated.

The full process iteratively looks like this.

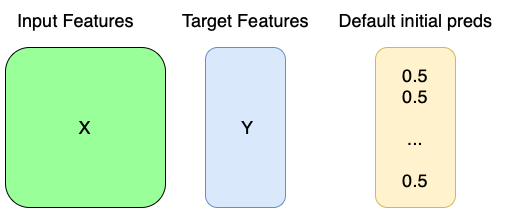


Thats the end of regression in XGboost (at least the main chunk)

**Classification**

For simplicity lets initially consider that the problem statement is that of binary classification.

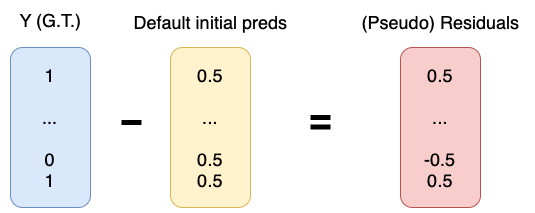
**I. Given data and initial predictions**



This remains the same, except you Y values will be discrete and not continuous.

**II. Calculating Pseudo Residuals**

This step pretty much remain the same.



Since Y is discrete this changes slightly but its the same concept.

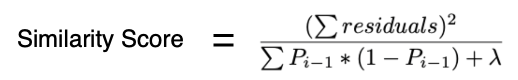
**III. Building XGBoost Trees (main part of algo.)**

1. **Start with all residuals in the same leaf.**(Same as Regression)



All residuals are in the same leaf/node.

**2. Calculate similarity score**Like we did in regression, we will calculate similarity score for nodes. Although the formula changes slightly for classification.



In the denominator instead of number of residuals be have a sum of previous probability equations.

**3. Finding the best splitting feature and its values.***(Same as regression)*

Go through all of the features one by one. Then sort all values in a feature and go through the values one by one. Find the mean of 2 values at a time and split leaf values according to that value.

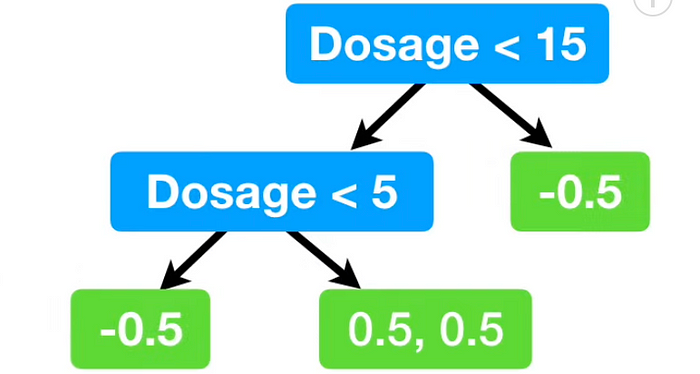
**4. Calculating Gain***(Same as regression)*

Find the split with the highest gain.



**5. Creating the tree**

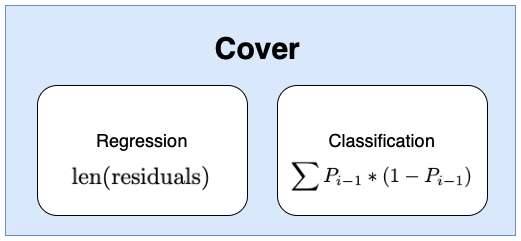
Keep splitting the nodes in your nodes according to steps 3,4 to build the tree.



After this you may get a tree that looks something like this.

The stopping conditions are either the max depth has been reached (6 by default) or the leaf has minimum number of residual in it (calculated using cover).

What is ‘cover’?  
Cover is basically defined as the denominator of similarity score minus lambda. So basically the formula is…



Formulae for Cover in Regression and classification.

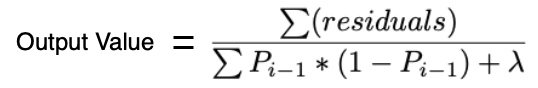
By default the value of cover is set to 1 in both cases.  
***Regression:*** Since the minimum value of cover is 1 by default and we cant have less that 1 residuals in a leaf cover has no effect on how we grow the tree for the default value of cover.  
***Classification:*** If the calculated cover value for a leaf is less than 1 then we discard that division (essentially pruning the left).

Note — Cover in code is the min\_child\_weight parameter.

**6. Pruning the tree**(Same as regression)  
Start bottom up and just prune based on the hyper-parameter gamma (r) and gain for each node.

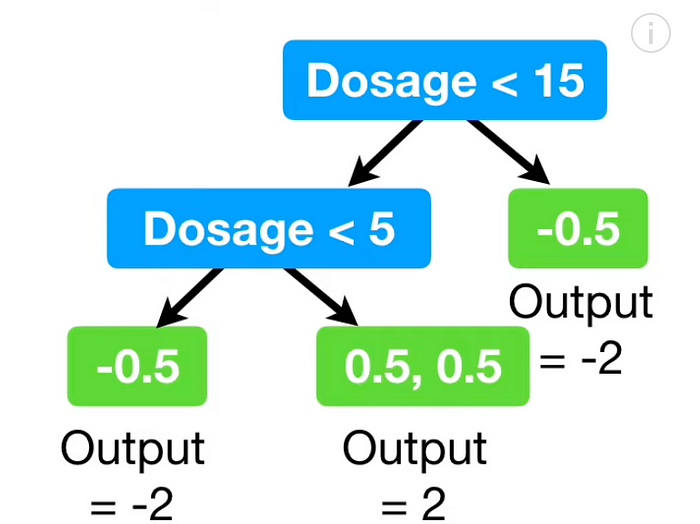
**7. Calculating Output**

Calculating the output value at each node.  
Heres the formula. As you can see its quite similar to what we saw in Regression.



Output formula for each node (Classification)

Using the above formula you can calculate the output values as each leaf node and you’ll get some thing like this

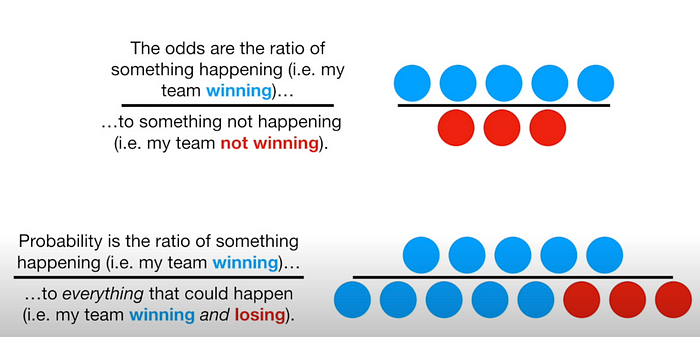


Sample tree with output values calculated.

Finally we have our first tree!!!

**IV. Updating the residuals and getting the output**

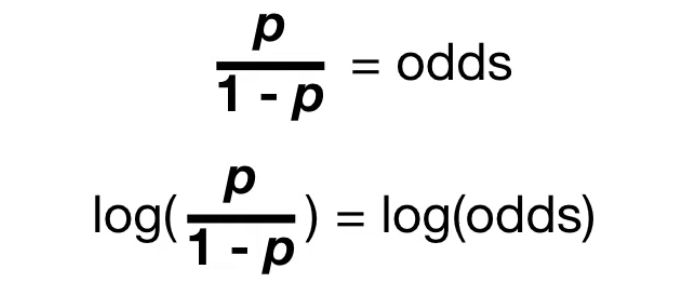
But wait this isn’t regression and we cant simply add these values (multiplied by learning rate(eta)). We need to convert the initial probability (0.5) into log(odds), perform the addition with lr multiplied and convert it back to probability. Let’s take it step by step.



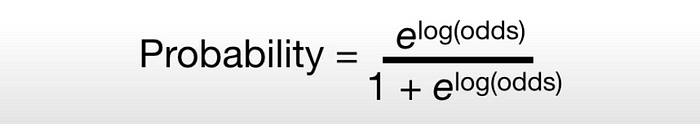
Quick overview of the difference between odds and probability (Context — if my team wins 5 out of 8 games). Difference between odds(top) and probability(bottom)

So now you know odds and probability? If still unclear watch [this](https://www.youtube.com/watch?v=ARfXDSkQf1Y) and then continue.

How do I convert odds to probability?



Formula to calculate odds from probability.



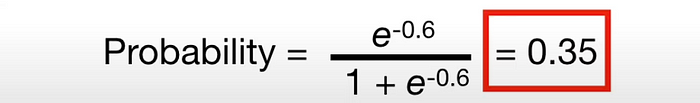
Do some math and you’ll this equation as well.

Okay now we have all of the tools and understanding necessary for the next step.

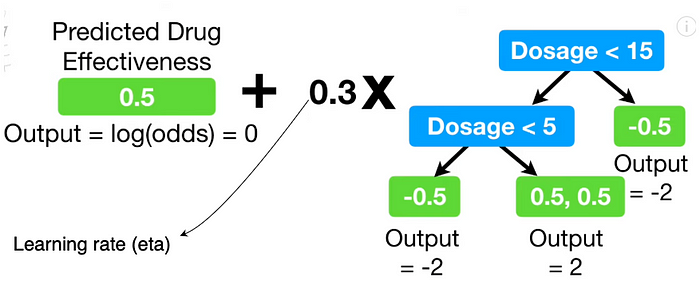
1. Convert the default initial probability (0.5) to log(odds). So that is, log(odds) = log(p/(1-p)) = log(0.5/1–0.5) = log(0.5/0.5) = log(1) = 0
2. Perform the step of adding the node value multiplied by lr



3. Convert this log(odds) value back to probability using the second formula.

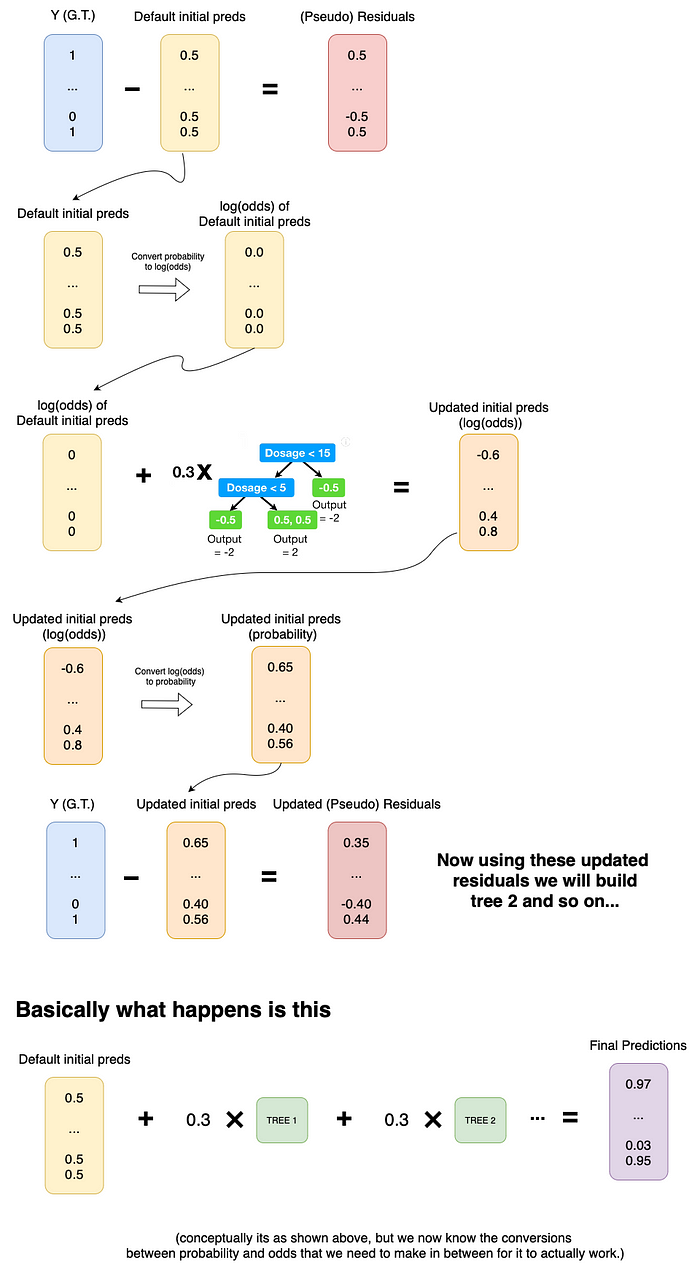


Thats all! Thats one iteration for classification!



How the initial preds are updated.

The full process iteratively looks like this.

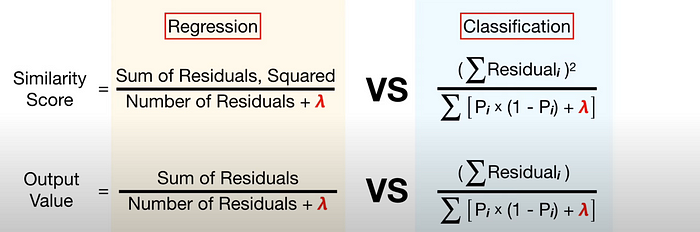


Thats the end of Classification in XGboost (at least the main chunk)

**Mathematical Details!!!**

You may have noticed that we just conjured most of the formulae for calculating stuff like similarity score and output. I’ll be diving into proofs (and hence reasons) for why they are the way they are.

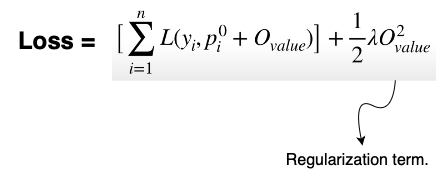
This section is skippable, but the math isn’t complicated so I’d suggest at least skim over it.



The formulae we’ll be deriving

Lets begin…

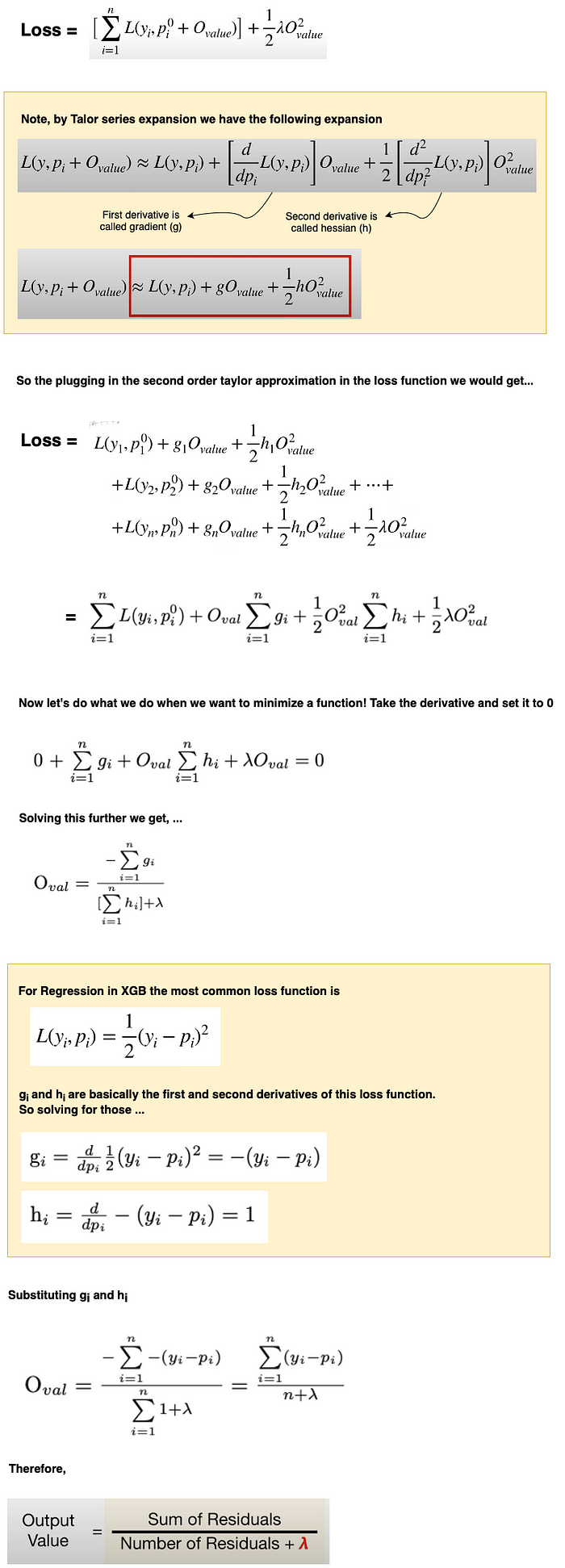
We can determine the performance of our XGB models using loss functions. The general structure of loss function is as follows.



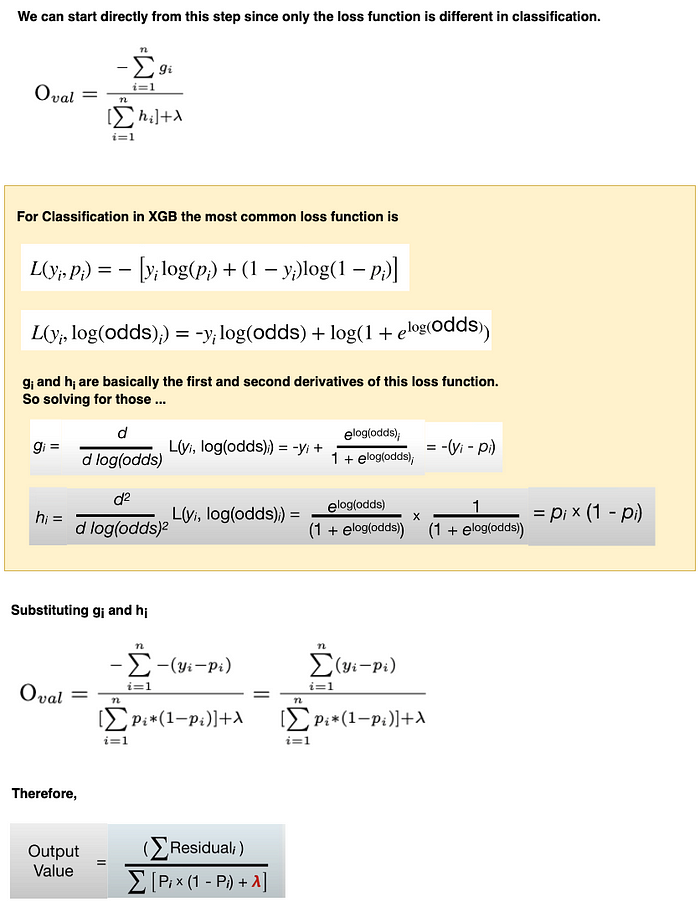
Full Loss function (generic)

**Okay, now lets derive the output value formula first**

Regression first.



Now to derive the output value formula for classification.



Output value formula derivation for Regression

**Now let’s derive the similarity score formula!**

🚧 work in progress 🚧

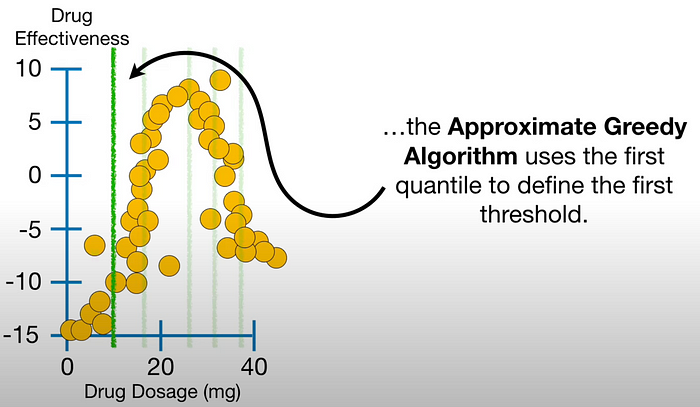
**Crazy Cool optimizations**

**Approximate Greedy Algorithm**

XGBoost uses a greedy algorithm to build trees because we split a node only based on the gain value and not how that particular split will affect the splitting in the future.

But even so, the greedy algorithm becomes slow because we need to check gain values for every split in every feature. This is where the approximate greedy algorithm comes in.

Instead of taking every pair of values in a feature and calculating the gain, we can instead divide the feature values in quantiles and test only for those values.



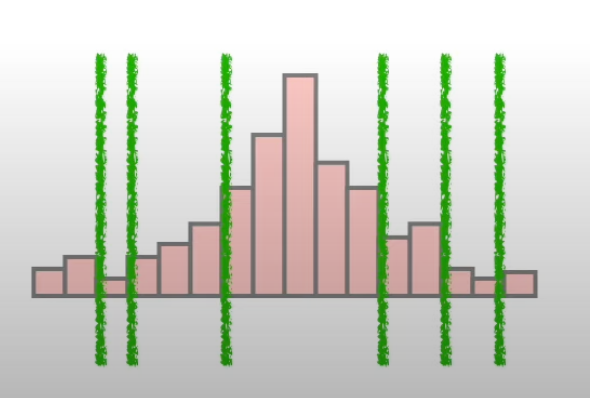
Approximate Greedy Algorithm — quantile division (visually)

But..

How many quantiles should we split it by? Less is quick but inaccurate. More is slow but accurate. The default value is *about* 33 quantiles.

We have “about” 33 quantiles because these are not ordinary quantiles. We useparallel learning and sketch algorithm to get the weighted quantiles for a feature. Explaining this would take very long so those interested can have a look [here](https://medium.com/@wjj1019/in-depth-overview-of-xgboost-partii-45384b90d818).

The main crux of it is the data is split by weighted quantiles, that is, every quantile wont necessarily have the same number of samples. It may look somthing like this…



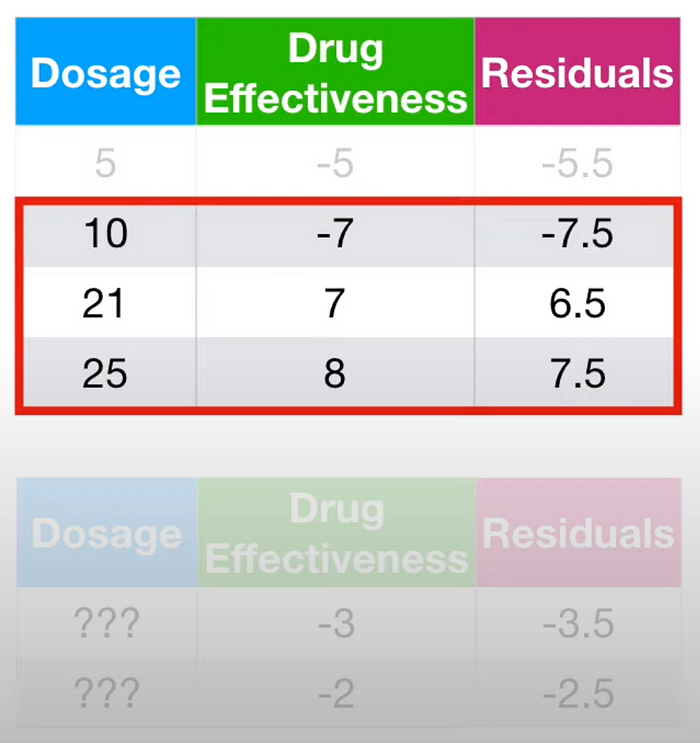
weighted quantiles

Note — This is only used when the dataset is huge. If the dataset is small XGBoost just uses the normal greedy algorithm.

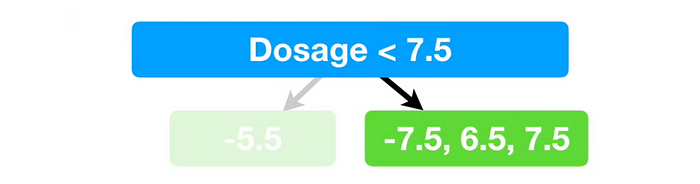
**Sparsity-aware split finding**

Basically, how does XGBoost handle missing values inherently.

Lets say we have a feature ‘Dosage’ with some nan values. We split the rows in 2 tables. One table without nan values and one with nan values.



Now



**Cache aware access**

Gradients and Hessians are stored in Cache. Thats pretty much it lol.

**Questions**

1. What happens to categorical features when we care trying to find the best splitting values in features?  
   Convert to one hot encoding. But the lastest version has released a more sophisticated way in beta. (need to understand that.)  
   — or label encoding
2. How does it work for multi-class classification?  
   One-v-one or one-v-rest approach is used.
3. Similarity score formula intuition?

**References**

I learnt all of it from [Statquest](https://www.youtube.com/watch?v=OtD8wVaFm6E) and quite honestly watching his videos is much better than this blog. So I’d suggest use this for revision and not first-hand learning. I’ve also taken a few screenshots and edited them from his videos, so yeah, there.

[Xgboost](https://medium.com/tag/xgboost?source=post_page-----e1cae7c5b6cb---------------xgboost-----------------)

[Machine Learning](https://medium.com/tag/machine-learning?source=post_page-----e1cae7c5b6cb---------------machine_learning-----------------)

[AI](https://medium.com/tag/ai?source=post_page-----e1cae7c5b6cb---------------ai-----------------)