XG Boost

03 August 2021 10:50

Introduction

It's is also known as ***"Extreme Gradient Boosting"***

XGboost is the most widely used algorithm in machine learning, whether the problem is a classification or a regression problem. It is known for its good performance as compared to all other machine learning algorithms.

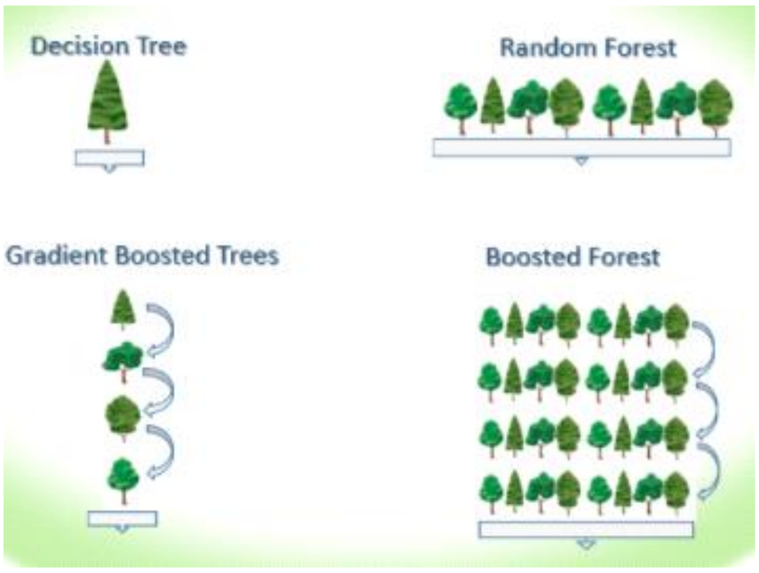
Even when it comes to machine learning competitions and hackathon, XGboost is one of the excellent algorithms that is picked initially for structured data. It has proved its determination in terms of **speed and performance**.

"It is the execution of gradient boosted decision trees that is designed for high speed and performance."

Decision Tree - based algorithms are considered to be good performers when it comes to **small to medium structured data or tabular data. XGboost** is commonly used for **supervised learning** in **machine learning**.

==

Before understanding of XG Boost, we need to clear in understanding of what is a **Decision Tree** and What are **Boosting methods**?



Boosting is nothing but ensemble techniques where previous model errors are resolved in the new models. These models are added straight until no other improvement is seen.

Gradient boosting is a method where the new models are created that computes the error in the previous model and then leftover is added to make the final prediction.

Similarly, XG Boost which is called as Xtreme Gradient Boosting method which is also works like Gradient boost method Which was designed to be used with large, complicated data sets.

Maths behind XGBoost algorithm explained with Data Step by Step

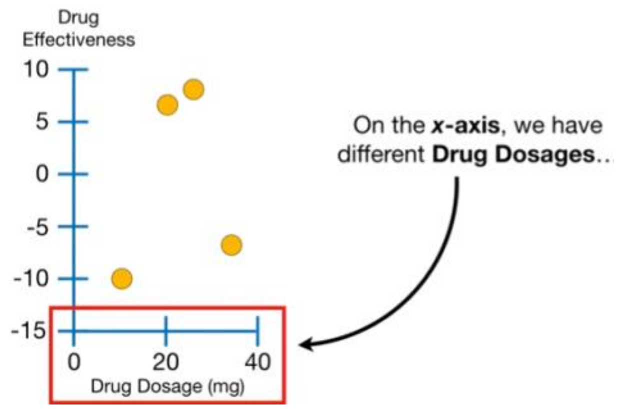
==

STEPS FOR XGBOOST:

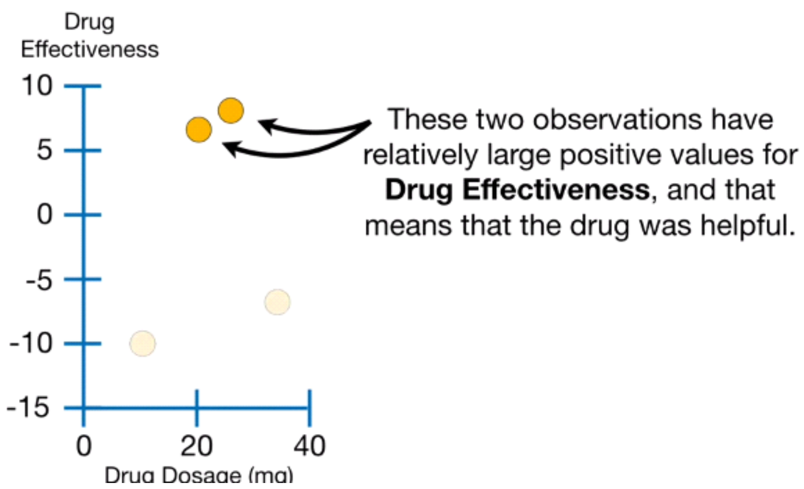
Step1. Consider initial prediction 0.5
Step2. Split the tree based on independent features just like our regular decision tree
Step3. calculate similarity weigh for each leave **similarity score** = (Σ Residuals)2/(Total residuals+ λ)
if you are using classification then
similarity score = (Σ Residuals)2/([Previous Probability * (1 - previous probability)]+ λ)
Step4. calculate **Gain =** Left Similarity score + Right Similarity score - Root Similarity.
Step5. Based on the highest Gain value tree will start expanding itself.
Step6. For Pruning our tree we will introduced Gamma (eta) values.
Step7. Calculate output value = sum of residuals / (Total residuals + λ)
Step8. New predictive value = Initial value (Y_pred (0.5)) + **Learning Rate** * Output value of node

==

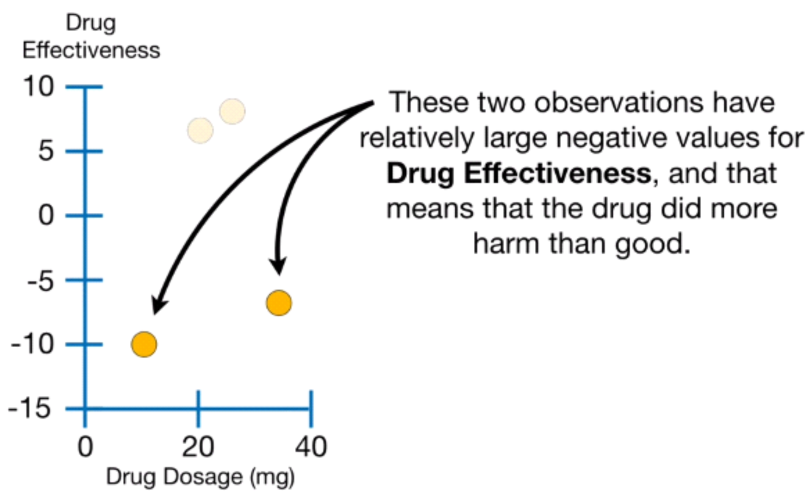
Let's look at how XGboost works with an example. Here I'll try to predict a Drug effectiveness based on Drug Dosage.



==



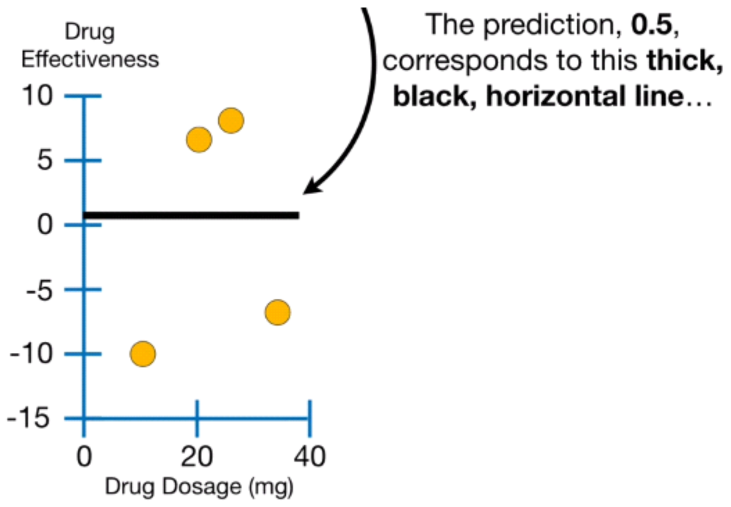
==



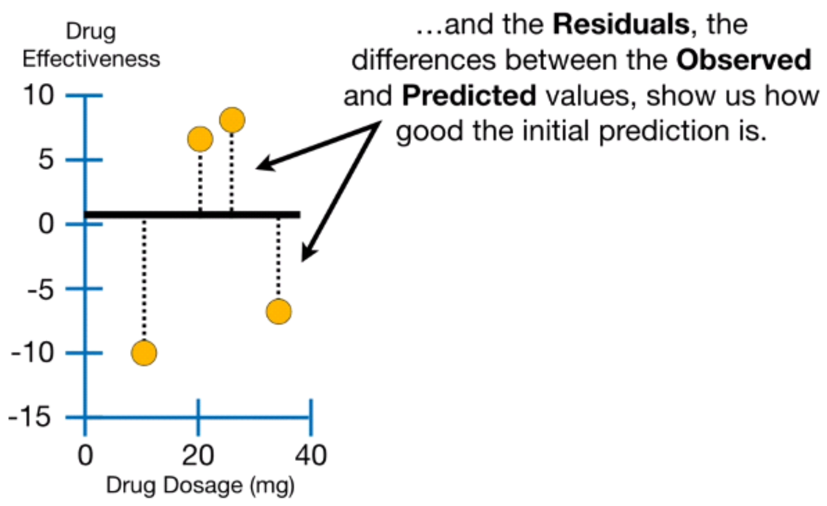
==

- The first step in fitting XG Boost to the Training Data is to make an initial prediction
- This prediction can be anything, but by default it is 0.5, whatever it is either Classification or Regression.

==



==



Look out the differences between predicted line 0.5 to the actual data points those are residuals, Just like Gradient Boosting , **XG Boost** also fits a **regression tree** model on these **residuals**.

==

Let's note down what we receive the residuals from the graph above, Let's say those are

Residuals received from the first prediction = [-10.5, 6.5, 7.5, -7.5]

Now we calculate the Similarity score for the above residuals, Formulae for similarity score is given below.

$$\text{Similarity Score} = \frac{\text{Sum of Residuals, Squared}}{\text{Number of Residuals} + \lambda}$$

$$\lambda \; is\; a\; regularization\; parameter$$

==

If we insert $\lambda = 0$

$$\text{Similarity Score} = \frac{(-10.5 + 6.5 + 7.5 + -7.5)^2}{4 + 0}$$

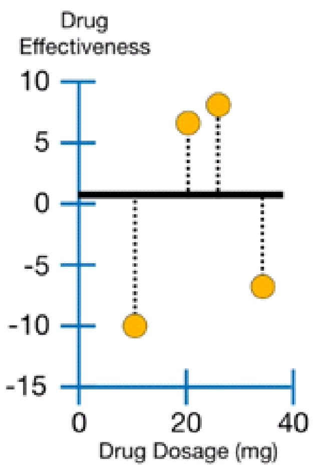
…and since there are **4 Residuals** in the leaf, we put a **4** in the denominator.

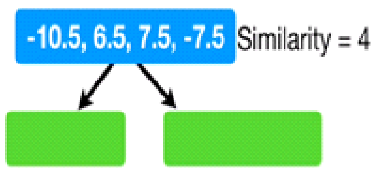
If we calculate it we will receive the value of Similarity Score = 4 for $\lambda = 0$

==

Now here each tree start with a single leaf, and we need to choose for splitting criteria for given X variable which is continuous

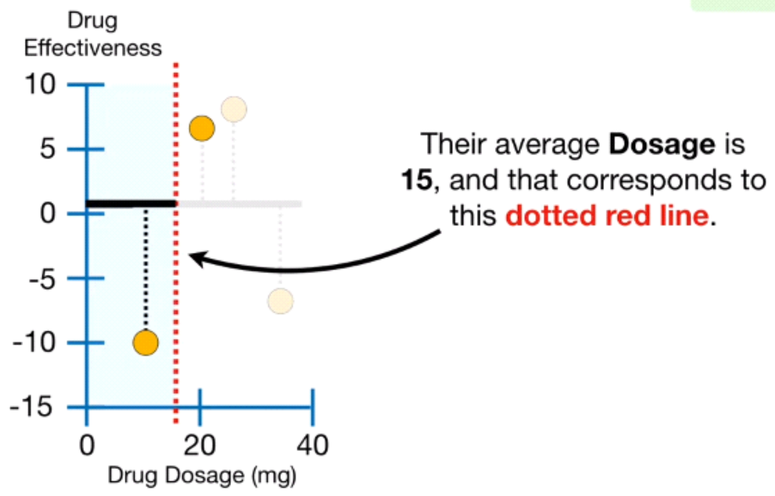
[-10.5, 6.5, 7.5, -7.5]





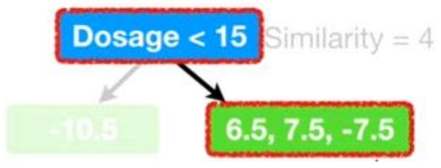
Now the challenge is while splitting them in to two groups, how better our sample points are getting splitting in to two groups.

==
To select that criteria, we will select first two rows from the data and will take its average dose from x values



If I choose as Dosage < 15, there will be only one point will be left and rest of the points will be on the right as below.

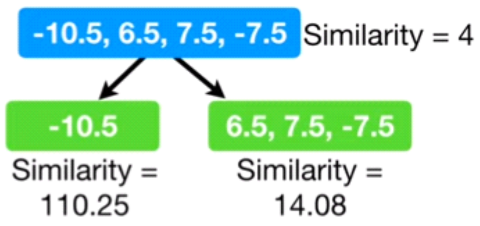
[-10.5, 6.5, 7.5, -7.5]



Now, we has to calculate the similarity score for the left the leaf and similarity score for the right of the leaf.

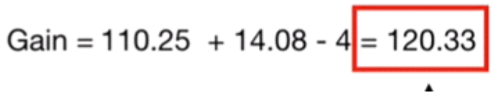
For

$$\text{Similarity Score} = \frac{-10.5^2}{1 + 0} \qquad \text{Similarity Score} = \frac{(6.5 + 7.5 + -7.5)^2}{3 + \lambda}$$

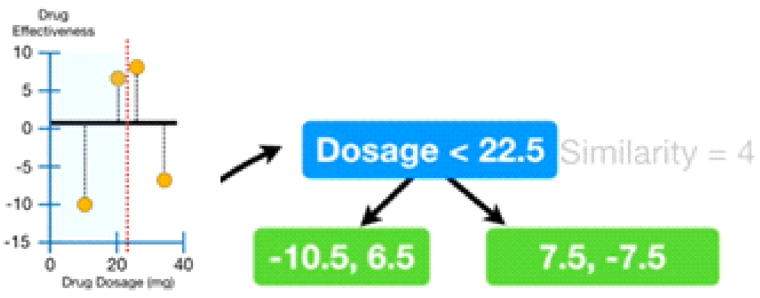


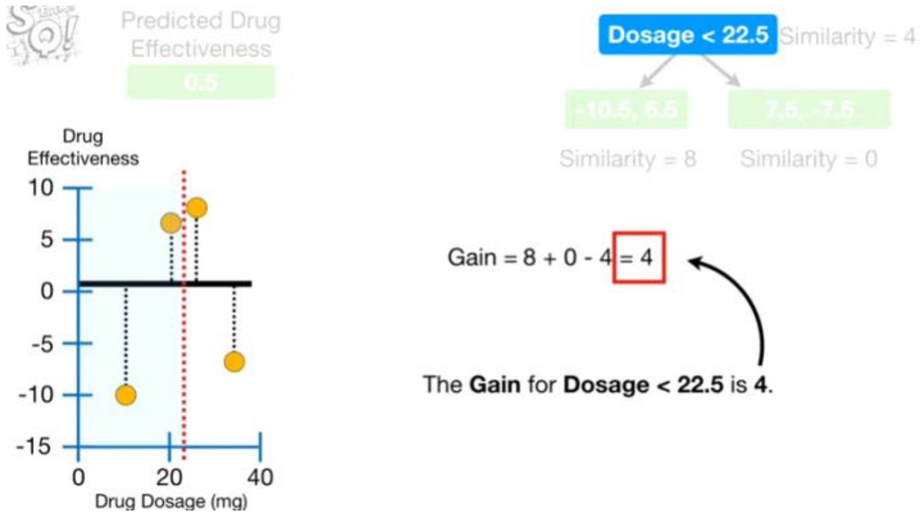
Now we need to quantify that how much these cluster leaves are better than the root residuals leave. We do this by calculating the Residuals in to two groups.

Gain = Left Similarity score + Right Similarity score - Root Similarity

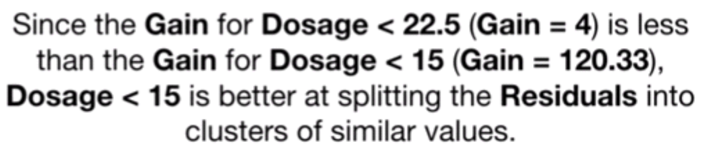

Gain = 110.25 + 14.08 - 4 = 120.33

==
Now this Gain value is the only value when we have taken average of first two samples i.e **Dosage < 15**, similarly we also need to calculate for other thresholds.





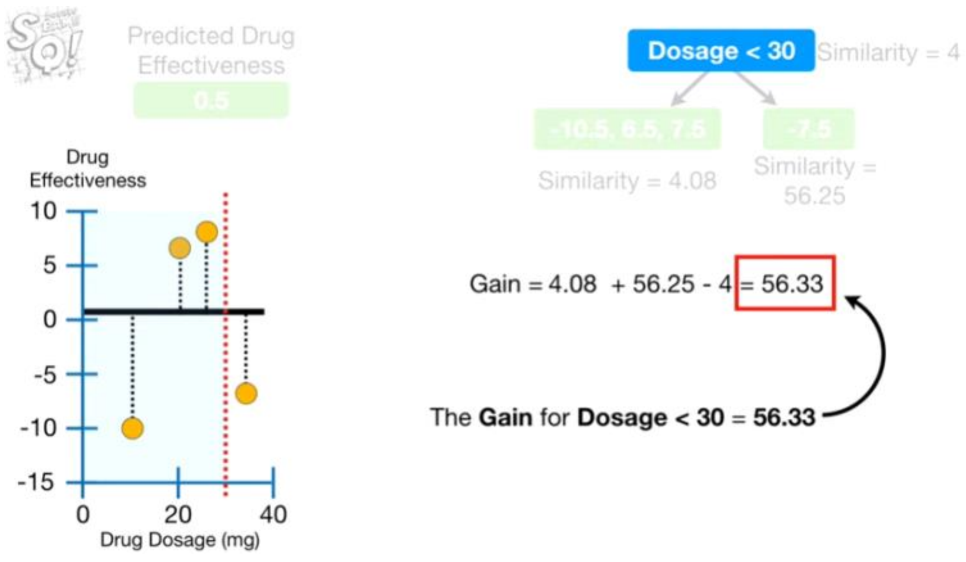
Note that now,

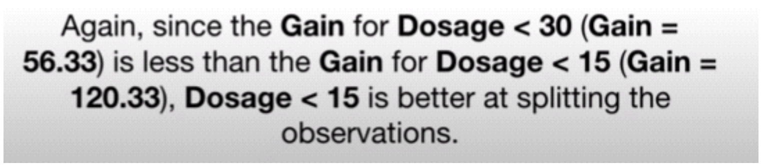


Since the **Gain** for **Dosage < 22.5** (**Gain = 4**) is less than the **Gain** for **Dosage < 15** (**Gain = 120.33**), **Dosage < 15** is better at splitting the **Residuals** into clusters of similar values.

==

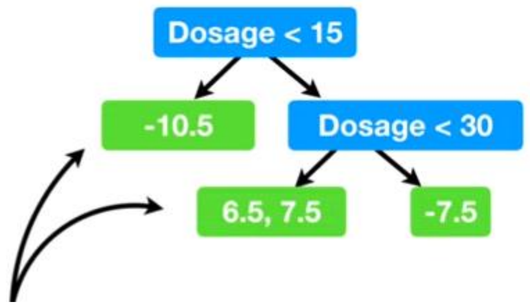
Now, let's do the left over threshold for last two rows and we get dosage < 30




Again, since the **Gain** for **Dosage < 30** (**Gain = 56.33**) is less than the **Gain** for **Dosage < 15** (**Gain = 120.33**), **Dosage < 15** is better at splitting the observations.

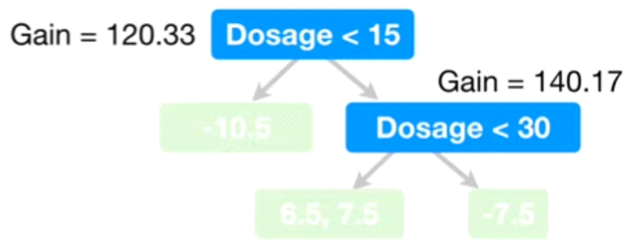
==

Here is our first leaf with Dosage < 15 will gets started and further nodes will be added till last.



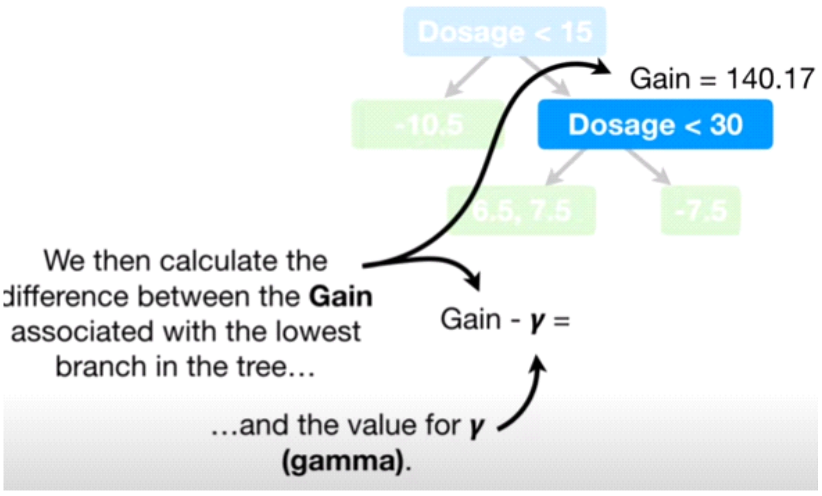
==

It’s a time for **Prune** our tree again to reduce the complexity.
We prune **XG boost** Tree based on its **Gain values.**



Here we are going to introduce another tuning parameter called **"Gamma".**

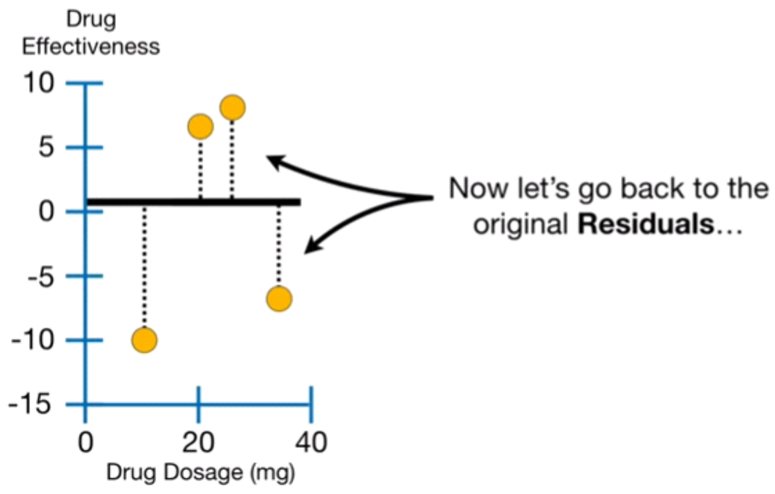
So, let's take gamma value a **130** which is a random value


Dosage < 15
Gain = 140.17
-10.5
Dosage < 30
6.5, 7.5
-7.5
We then calculate the difference between the **Gain** associated with the lowest branch in the tree…
Gain - γ =
…and the value for γ **(gamma)**.

If the difference between (**Gain - Gamma)** is **negative** we will remove the branch.
If the difference between (**Gain - Gamma)** is **Positive** we will continue using the branch.

If your difference is negative if lowest branch is removed , then it will check the root variable also, if it is also removed then your tree will be comeback to your normal position where your tree is initiated with errors are target.

===

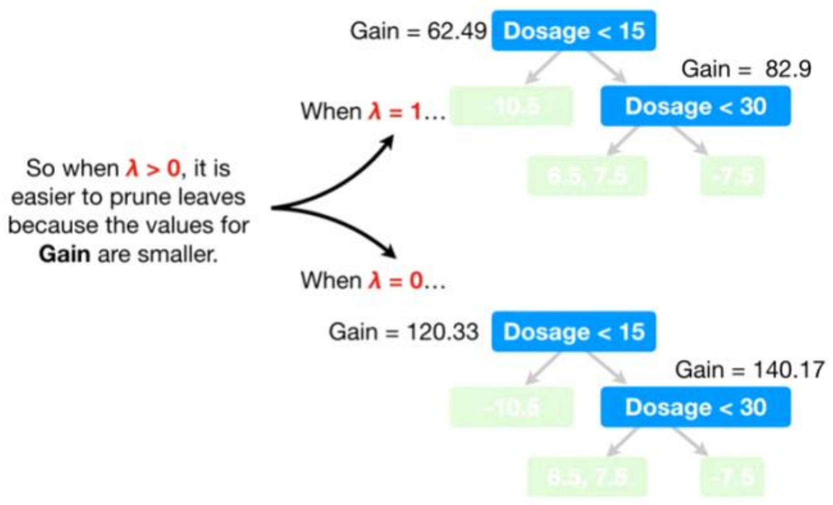


Now we are doing this process repeating again with Lambda value = 1

$$\text{Similarity Score} = \frac{\text{Sum of Residuals, Squared}}{\text{Number of Residuals} + 1}$$

Remember λ **(lambda)** is a **Regularization Parameter**, which means that it is intended to reduce the prediction's sensitivity to individual observations.

==



Here, to reduce the overfitting of the model we need to observe the lambda value closely.

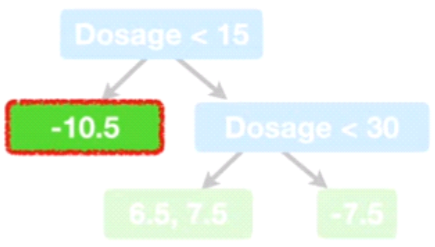
==

Now to decide for the given lambda and gamma value we has to observe the output value

Formulae for the output value

$$\text{Output Value} = \frac{\text{Sum of Residuals}}{\text{Number of Residuals} + \lambda}$$

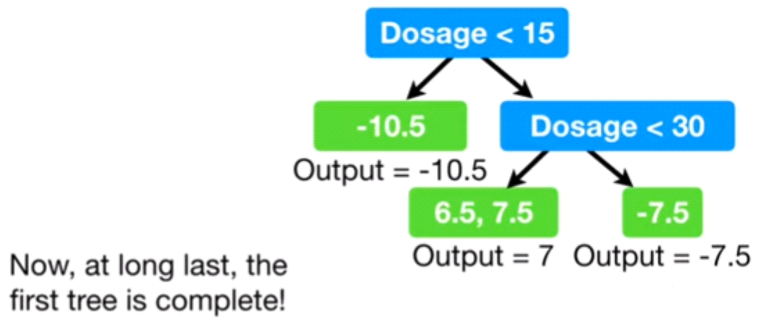
Example:



If $\lambda = 0$, then there is no **Regularization** and the **Output Value = -10.5.**

$$\text{Output Value} = \frac{-10.5}{1 + 0} = -10.5$$

If I calculate for all nodes with output value formulae it will exists as below values



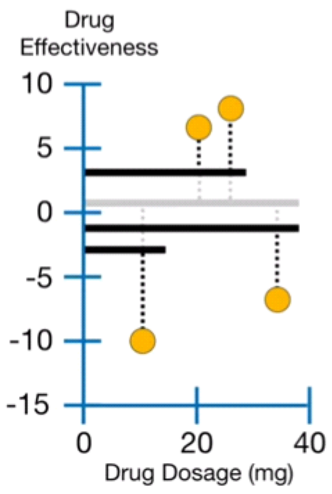
Since, we have constructed our first tree, we need to make new predictions.
==

Final step for predicting responses, we will add a new parameter called **Learning rate** (default value is 0.3)

Initial value (Y_pred = 0.5) + Learning Rate * Output value of node

1. **0.5 + (0.3 * -10.5) =-2.65**
2. **0.5 + (0.3 * 7)   = 2.6**
3. **0.5 + (0.3 * 7)   = 2.6**
4. **0.5 + (0.3 * -7.5) = -1.75**

Y_pred1 = [-10.5, 6.5, 7.5, -7.5]
Y_pred2 = [2.65, 2.6, 2.6, -1.75]



The black color lines are new predicted points(Y_pred2) which is less error than your earlier error (Y_pred1)

Now we will build another tree based on the current residuals and makes new predictions than previous residuals. And builds another tree based on the current residuals. And we keep building trees until all your residuals become smaller in size or we reached the maximum number.

==

Advantages:

XGBoost is an efficient and easy to use algorithm which delivers high performance and accuracy as compared to other algorithms. XGBoost is also known as regularized version of GBM. Let see some of the advantages of XGBoost algorithm:

1. **Regularization**: XGBoost has in-built L1 (Lasso Regression) and L2 (Ridge Regression) regularization which prevents the model from overfitting. That is why, XGBoost is also called regularized form of GBM (Gradient Boosting Machine).

While using Scikit Learn library, we pass two hyper-parameters (alpha and lambda) to XGBoost related to regularization. alpha is used for L1 regularization and lambda is used for L2 regularization.

2. **Parallel Processing**: XGBoost utilizes the power of parallel processing and that is why it is much faster than GBM. It uses multiple CPU cores to execute the model.

While using Scikit Learn library, **nthread** hyper-parameter is used for parallel processing. **nthread** represents number of CPU cores to be used. If you want to use all the available cores, don't mention any value for nthread and the algorithm will detect automatically.

3. **Handling Missing Values:** XGBoost has an in-built capability to handle missing values. When XGBoost encounters a missing value at a node, it tries both the left and right hand split and learns the way leading to higher loss for each node. It then does the same when working on the testing data.

4. **Cross Validation:** XGBoost allows user to run a cross-validation at each iteration of the boosting process and thus it is easy to get the exact optimum number of boosting iterations in a single run. This is unlike GBM where we have to run a grid-search and only a limited values can be tested.

5. **Effective Tree Pruning**: A GBM would stop splitting a node when it encounters a negative loss in the split. Thus it is more of a greedy algorithm. XGBoost on the other hand make splits up to the max_depth specified and then start pruning the tree backwards and remove splits beyond which there is no positive gain.

==

Parameters to tune in XGBoost:

- thread
- eta
- min_child_weight
- max_depth
- max_depth
- max_leaf_nodes
- gamma
- subsample
- colsample_bytree

For Further regarding, here is the official documentation page.

https://xgboost.readthedocs.io/en/latest/?ref=hackernoon.com