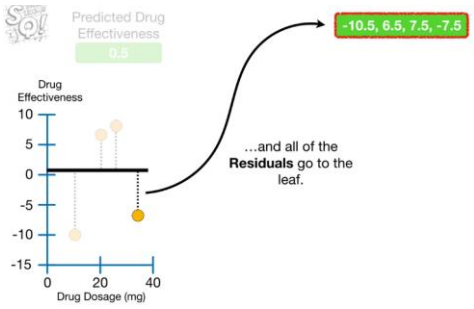
XGBoost for Regression

在 XGBoost 里第一个步骤就是初始一个预测值，Default Initialize Value 是 0.5，可以通过设置改变这个初始值。初始化后，XGBoost 会开始创建自己的树，这个树也叫 XGBoost Tree。



Output Value Equation Derive

$$Total\ Loss\ Function = \left[\sum_{i=1}^{n} L(y_i, p_i)\right] + \gamma T + \frac{1}{2}\lambda O_{value}^2$$

$$\frac{1}{2}\lambda O_{value}^2 \rightarrow Regularization$$

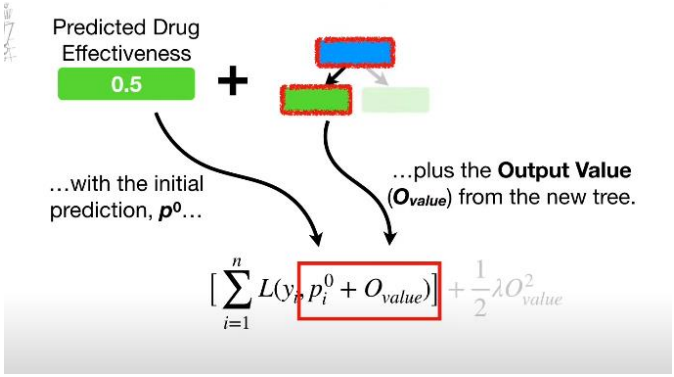
$$\gamma \rightarrow User\ Defineable\ Penalty\ (Pruning)$$

$$T \rightarrow Number\ of\ Terminal\ Node\ (Leaves)$$

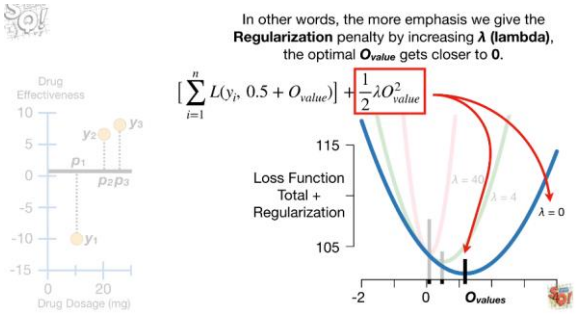
在这里可以忽略 γT 因为计算 $\gamma = 0$，还是能够达到 Pruning 的效果，Pruning 的步骤是在树建立完成之后执行，可是 γT 不会影响推算出 Optimal Output Values 或者 Similarity Score。

$$Total\ Loss\ Function = \left[\sum_{i=1}^{n} L(y_i, p_i)\right] + \frac{1}{2}\lambda O_{value}^2$$

目的是为了找出一个 O_{value} 可以让这整个 Equation 的值是最小的。



在这里以第一棵树作为列子，p_i^0 代表着第一片叶子，O_{value} 代表着第一棵树的 Output Value。



如上图，当 λ 的值越大，Optimal Value 就越靠近 0。

$$L(y_i, p_i + O_{value}) \approx L(y_i, p_i) + \left[\frac{d}{dp_i} L(y_i, p_i)\right] O_{value} + \frac{1}{2}\left[\frac{d^2}{dp_i^2} L(y_i, p_i)\right] O_{value}^2$$

$$L(y, p_i + O_{value}) \approx L(y, p_i) + gO_{value} + \frac{1}{2} hO_{value}^2$$

$$L(y, p_i) \rightarrow Loss\ Function\ for\ the\ Previous\ Prediction$$

$$Gradient \rightarrow g = \left[\frac{d}{dp_i} L(y, p_i)\right] \rightarrow First\ Derivative\ of\ Loss\ Function$$

$$Hessian \rightarrow h = \left[\frac{d^2}{dp_i^2} L(y, p_i)\right] \rightarrow Second\ Derivative\ of\ Loss\ Function$$

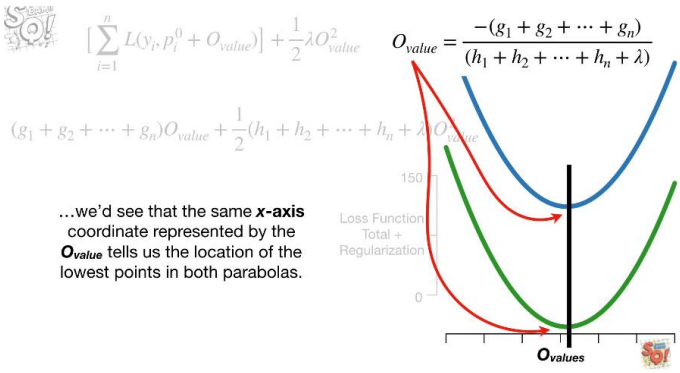
$$Total\ Loss\ Function = \left[\sum_{i=1}^{n} L(y_i, p_i)\right] + \frac{1}{2}\lambda O_{value}^2$$

$$Total\ Loss\ Function = \left[\sum_{i=1}^{n} L(y_i, p_i) + gO_{value} + \frac{1}{2} hO_{value}^2\right] + \frac{1}{2}\lambda O_{value}^2$$

在这里做的其实就是要 Minimize with a Second Order Taylor Polynomial。

$$L(y_1, p_1^0) + g_1 O_{value} + \frac{1}{2} h_1 O_{value}^2 + \cdots + L(y_n, p_n^0) + g_n O_{value} + \frac{1}{2} h_n O_{value}^2 + \frac{1}{2}\lambda O_{value}^2$$

在这里可以把所有的$L(y_i, p_i^0)$ 拿掉，因为这个不会影响到 O_{value}，当在做$\frac{d}{dO_{value}}$ 的时候，也会不见。



从上图可以看到，就算拿掉所有的 $L(y_i, p_i^0)$ 之后，两个 Graph 之间的中心点还是没变，只是 Scare 变了。

$$g_1 O_{value} + \frac{1}{2} h_1 O_{value}^2 + \cdots + g_n O_{value} + \frac{1}{2} h_n O_{value}^2 + \frac{1}{2} \lambda O_{value}^2$$

$$(g_1 + g_2 + \cdots + g_n) O_{value} + \frac{1}{2} (h_1 + h_2 + \cdots + h_n + \lambda) O_{value}^2$$

这时候可以就需要对这个 Equation 做 Derivative 然后让这个 Equation 等于 0，这是用来找出最低点的O_{value}。

$$\frac{d}{dO_{value}} (g_1 + g_2 + \cdots + g_n) O_{value} + \frac{1}{2} (h_1 + h_2 + \cdots + h_n + \lambda) O_{value}^2 = 0$$

$$(g_1 + g_2 + \cdots + g_n) + (h_1 + h_2 + \cdots + h_n + \lambda) O_{value} = 0$$

$$\boldsymbol{O_{value} = \frac{-(g_1 + g_2 + \cdots + g_n)}{(h_1 + h_2 + \cdots + h_n + \lambda)}}$$

$$L(y_i, p_i) = \frac{1}{2} (y_i - p_i)^2$$

$$g_i = \frac{d}{dp_i} \frac{1}{2} (y_i - p_i)^2 = -(y_i - p_i)$$

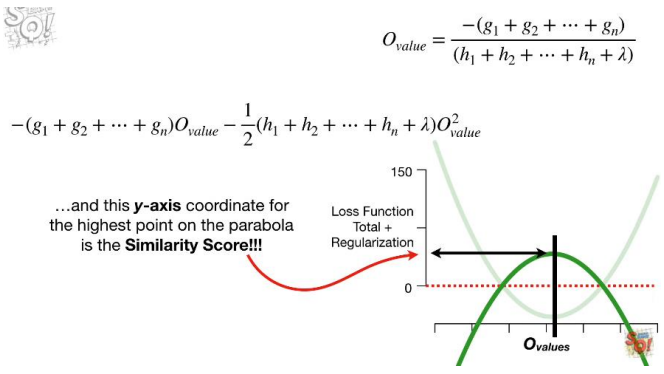
$$h_i = \frac{d^2}{dp_i^2} \frac{1}{2} (y_i - p_i)^2 = \frac{d}{dp_i} -(y_i - p_i) = 1$$

$$O_{value} = \frac{-\big(-(y_1 - p_1) + -(y_2 - p_2) + \cdots + -(y_n - p_n)\big)}{(1 + 1 + \cdots + 1 + \lambda)}$$

$$O_{value} = \frac{\big((y_1 - p_1) + (y_2 - p_2) + \cdots + (y_n - p_n)\big)}{Number\ of\ Residuals + \lambda}$$

$$\boldsymbol{O_{value}\ for\ a\ Leaf = \frac{Sum\ of\ Residuals}{Number\ of\ Residuals + \lambda}}$$

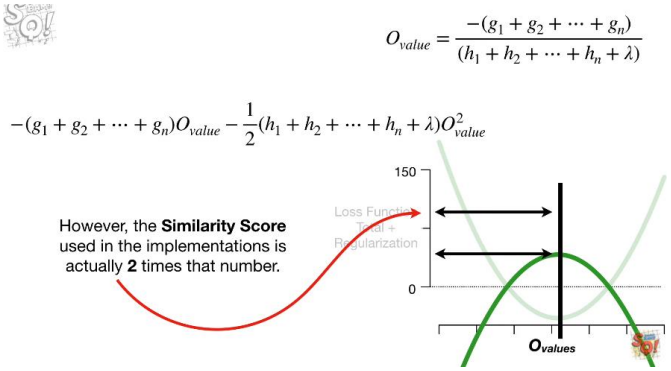
Similarity Score Equation Derive



$$(g_1 + g_2 + \cdots + g_n) O_{value} + \frac{1}{2} (h_1 + h_2 + \cdots + h_n + \lambda) O_{value}^2$$

把原来的 Taylor Series Polynomial 后的 Total Loss Function 乘上-1，这个 Equation 的 Graph 就会反转，而在这个最顶点的位置就是 Similarity Score。

$$-(g_1 + g_2 + \cdots + g_n)O_{value} - \frac{1}{2}(h_1 + h_2 + \cdots + h_n + \lambda)O_{value}^2$$



但是这个 Similarity Score 用在 Implementation 里是比上面这个 Equation 算出的多一倍。

$$O_{value} = \frac{-(g_1 + g_2 + \cdots + g_n)}{(h_1 + h_2 + \cdots + h_n + \lambda)}$$

$$-(g_1 + g_2 + \cdots + g_n)\frac{-(g_1 + g_2 + \cdots + g_n)}{(h_1 + h_2 + \cdots + h_n + \lambda)}$$
$$-\frac{1}{2}(h_1 + h_2 + \cdots + h_n + \lambda)\frac{-(g_1 + g_2 + \cdots + g_n)}{(h_1 + h_2 + \cdots + h_n + \lambda)}$$

$$\frac{(g_1 + g_2 + \cdots + g_n)^2}{(h_1 + h_2 + \cdots + h_n + \lambda)} - \frac{1}{2}(h_1 + h_2 + \cdots + h_n + \lambda)\left[\frac{-(g_1 + g_2 + \cdots + g_n)}{(h_1 + h_2 + \cdots + h_n + \lambda)}\right]^2$$

$$\frac{(g_1 + g_2 + \cdots + g_n)^2}{(h_1 + h_2 + \cdots + h_n + \lambda)} - \frac{1}{2}\frac{(g_1 + g_2 + \cdots + g_n)^2}{(h_1 + h_2 + \cdots + h_n + \lambda)}$$

$$Similarity\ Score = \frac{1}{2}\frac{(g_1 + g_2 + \cdots + g_n)^2}{(h_1 + h_2 + \cdots + h_n + \lambda)}$$

$$Similarity\ Score = \frac{(Sum\ of\ Residuals)^2}{Number\ of\ Residuals + \lambda} \rightarrow Omit\ \frac{1}{2}$$

如上面推算出的 Similarity Score Equation，乘上了一个 $\frac{1}{2}$，但是这个 $\frac{1}{2}$ 在 Implementation 的时候被拿掉了，因为 Similarity Score 只是相对测量，只要算出的每一个 Similarity Score 的 Scale 是一样的，也能得到同样的结果。

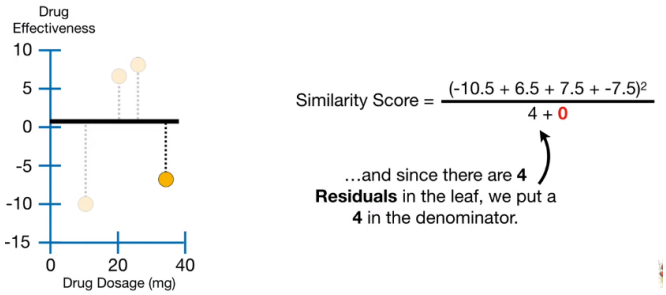
Create Tree

首先，XGBoost Tree 从一片叶子开始，所有的 Residuals 都在这片叶子里。之后就对这片叶子进行 Similarity Score 的计算。

$$Similarity\ Score = \frac{(Sum\ of\ Residuals)^2}{Number\ of\ Residuals + \ \lambda}$$

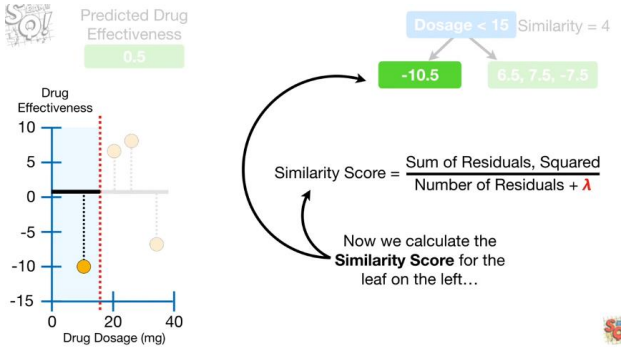
$$\lambda\ (Lambda\ ) \rightarrow Regularization\ Parameter$$

Similarity Score 越大，代表着这组数据里的的所有 Data Points 比较相似，Similarity Score 小代表着这组数据里的所有 Data Points 差别比较大。



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

如上图，这时候计算出的 Similarity Score 是 4。计算完第一片叶子之后，能够进行仔细的分类。



首先先对第一和第二个 Data 之间去平均值，如上图，第一个 Data Point 与第二个 Data Point 的平均是 15。小于 15 的 Data 只有一个，大于 15 的 Data 有 3 个。之后就能进行 Similarity Score 的计算。

$$Similarity\ Score\ for\ Left\ Leaf = \frac{(-10.5)^2}{1 + 0} = 110.25$$

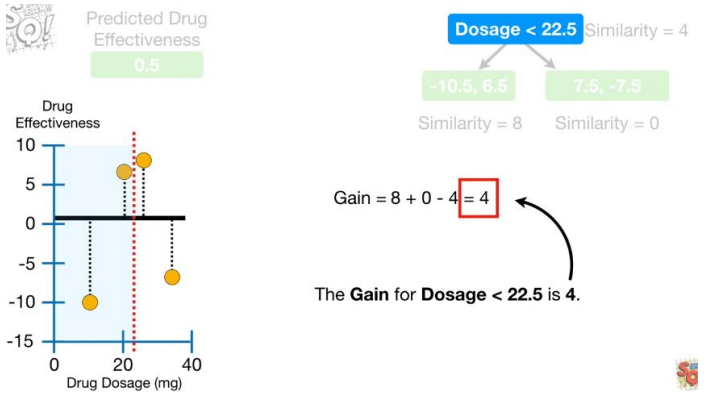
$$Similarity\ Score\ for\ Right\ Leaf = \frac{(6.5 + 7.5 - 7.5)^2}{3 + 0} = 14.08$$

当计算完左边叶子与右边叶子的 Similarity Score 之后，就得到了 3 个 Similarity Score，就需要计算 Gain。

$$Gain = Left_{Similarity} + Right_{Similarity} - Root_{Similarity}$$

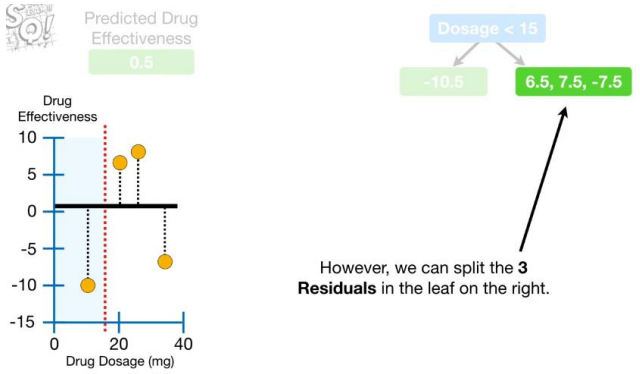
$$Gain\ for\ Dosage < 15 = 110.25 + 14.08 - 4 = 120.33$$

当第一个与第二个 Data Point 取平均当作 Threshold 的 Gain 计算完之后，就需要计算第二个 Data Point 与第三个 Data Point 取平均当 Threshold 的 Gain，一直重复直到到最后第二个与最有一个 Data Point 取平均当作 Threshold 的 Gain 计算完毕。

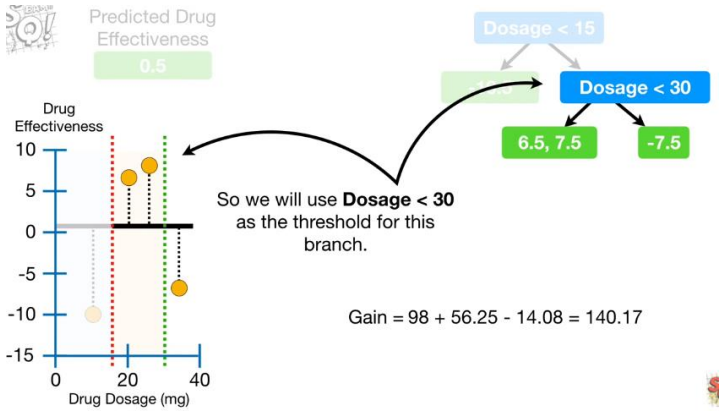


如上图是第二个与的第三个 Data Point 取平均后当作 Threshold 计算出来的 Gain，可以看出是比 $Dosage < 15$ 的 Gain 还要小很多，代表着$Dosage < 15$ 的 Threshold 比$Dosage < 22.5$ 的 Threshold 来的更好。

当所有 Threshold 的 Gain 都计算完之后，选出有着最大 Gain 的 Threshold 当作这棵树的一个 Branch。



如上图选出的最好的 Threshold 就是 $Dosage < 15$，这时候可以把叶子里的 Data Points 也进行分类，左叶只有一个值所以没办法 Further Split，但是右叶有 3 个 Data Points 所以能够继续 Split。也是使用同样的方法来计算 Gain，然后选出有着最大 Gain 的 Threshold。

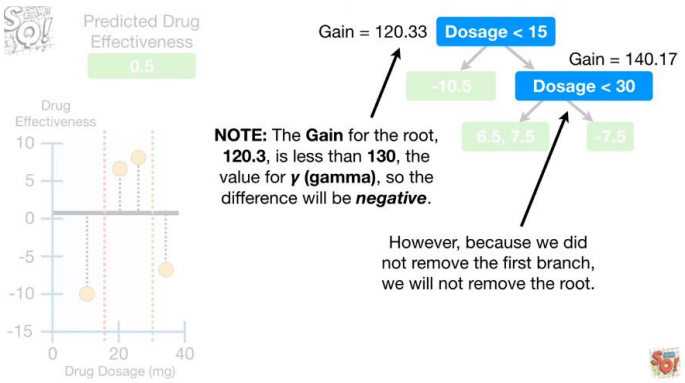


如上图，接入层计算出的 Threshold 是 $Dosage < 30$。在创建树的时候需要设置树的 Level，当已经达到树的 Level 之后，就不会再继续做 Further Splitting 的动作。在 XGBoost 里，XGBoost Tree 的 Default Level 是 6。

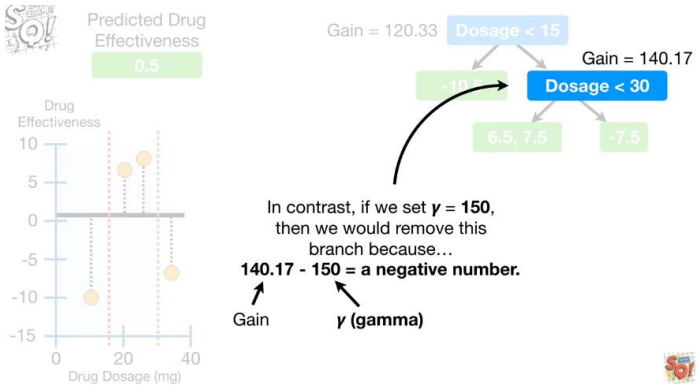
Prune Tree

在 XGBoost 里，要 Prune 一棵树是需要用到之前计算过的 Gain 的值。首先先设定一个 γ 值。

$$Prune\ Tree \rightarrow \begin{cases} Gain - \gamma < 0, & Prune\ Tree \\ Gain - \gamma \geq 0, & Dont\ Prune \end{cases}$$



在 XGBoost Tree 里，就算计算出的$Gain - \gamma < 0$， 只要 Child 没有被 Prune 掉，就会继续保留该 Branch，如上图。



当计算出的 $Gain - \gamma < 0$ 时，该 Branch 就会被 Prune，当整棵树都被 Prune 完之后，预测的结果就 Original Prediction，也就是还没创建该树的 Prediction 结果。除此之外，计算当$\gamma = 0$ 的时候，计算出的 Gain 小与 0 的情况，该 Branch 也会自动被 Prune 掉。In other words, setting $\gamma = 0$ does not turn off pruning.

在 Similarity Score 里，λ 的值越大，代表着计算出的 Similarity Score 也会变小，也是用来 Reduce Prediction's Sensitivity to Individual Observations。

$$Similarity\ Score = \frac{(-10.5)^2}{1+0} = 110.25$$

$$Similarity\ Score = \frac{(-10.5)^2}{1+1} = 55.12$$

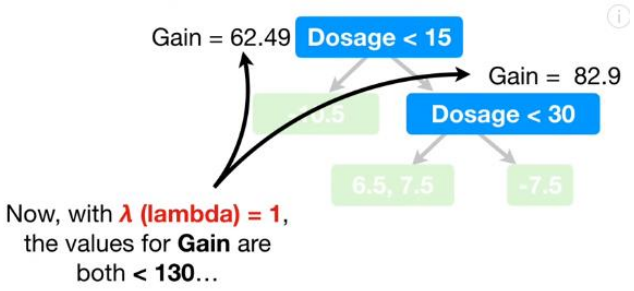
当只有一个 Residual 的时候，$\lambda > 0$ 的时候，比 $\lambda = 0$ 的时候降低了$\approx 50\%$。

$$Similarity\ Score = \frac{(6.5+7.5-7.5)^2}{3+0} = 14.08$$

$$Similarity\ Score = \frac{(6.5+7.5-7.5)^2}{3+1} = 10.56$$

当有 3 个 Residual 的时候，$\lambda > 0$ 的时候，比 $\lambda = 0$ 的时候降低了$\approx 28\%$。

当 Number of Residuals 越小，而 $\lambda > 0$ 的时候，Similarity Score Decrease 的 Ratio 越大，当 Number of Residuals 越大，而 $\lambda > 0$ 的时候，Similarity Score Decrease 的 Ratio 比较小。

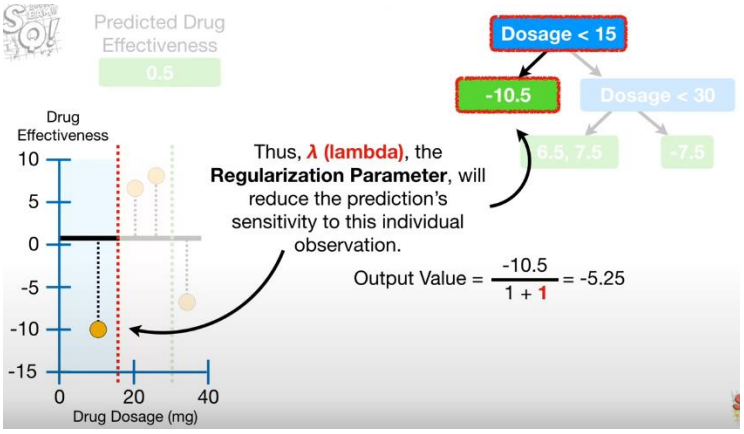


在之前没有被 Prune 掉的 Branch，当$\lambda > 0$ 的时候，很高几率会被 Prune 掉，这是因为 Gain 的值变小。$\lambda > 0$ 带来的好处是更容易 Prune 掉一些 Branch。On the other hand, by setting $\lambda > 0$, λ did what it was supposed to do; it prevented over fitting the training data.

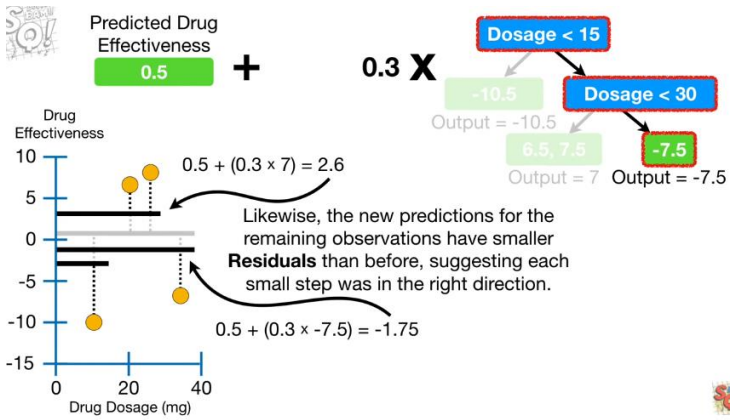
当树创建好，也计算完 Gain，也 Prune 掉一些 Branch 之后，需要计算每个叶子的 Output Value。

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

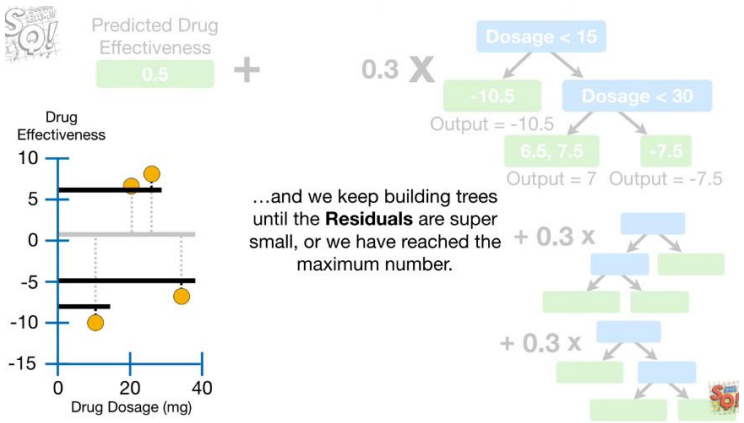
这个 Output Value 的 Equation 与 Similarity Score 的 Equation 很相似，除了取 Sum of Residual 的平方值。这里的 λ 值与 Similarity Score 的 λ 值是同样的。



当$\lambda = 0$的时候，Output Value 不会被 Regularize，当 $\lambda > 0$的时候，Output Value 会变小，也代表了这个 Output Value 被 Regularize 了。



当 Output Value for All Leaves 计算完毕之后，就能给出预测值。也就是原来的 Prediction 加上这棵树的 Output Value 乘上一个 Learning Rate。当第一颗树完成之后，就能计算出新的 Residuals，也就是取当前预测的值与 Actual Value 的差别做为新的 Residuals，然后再重复上面的步骤创建出新的树。

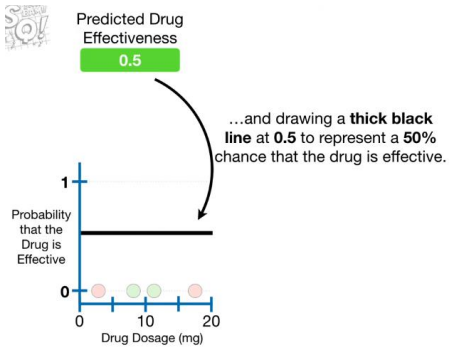


一直重复直到 Residuals 非常小，或者是到达了设定的树的数目。

$$Loss\ Function \rightarrow L(y_i, p_i) = \frac{1}{2}(y_i - p_i)^2$$

$$y_i \rightarrow Actual\ Value, p_i \rightarrow Prediction\ Value$$

XGBoost for Classification



XGBoost for Classification 也需要一个初始值，就是第一片叶子，通常是设置成 0.5，也可以通过计算后改动。之后就能创建第一颗 XGBoost Tree，这时候需要计算 Similarity Score 来选出适合的 Threshold。

Output Value Equation Derive

$$L(y_i, p_i) = -[y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]$$

$$L(y_i, \log(odds)_i) = -y_i \log(odds) + \log\left(1 + e^{\log(odds)}\right)$$

$$Gradient \rightarrow g_i = \frac{d}{dlog(odds)} L(y_i, \log(odds)_i) = -y_i + \frac{e^{\log(odds)_i}}{1 + e^{\log(odds)_i}} = -(y_i - p_i)$$

$$Hessian \rightarrow h_i = \frac{d^2}{dlog(odds)^2} L(y_i, \log(odds)_i) = \frac{e^{\log(odds)}}{1 + e^{\log(odds)}} * \frac{1}{1 + e^{\log(odds)}} = p_i * (1 - p_i)$$

$$\boldsymbol{O_{value}} = \frac{-(\boldsymbol{g_1} + \boldsymbol{g_2} + \cdots + \boldsymbol{g_n})}{(\boldsymbol{h_1} + \boldsymbol{h_2} + \cdots + \boldsymbol{h_n} + \boldsymbol{\lambda})}$$

$$O_{value} = \frac{-\left(-(y_1 - p_1) + -(y_2 - p_2) + \cdots + -(y_n - p_n)\right)}{(p_1 * (1 - p_1) + p_2 * (1 - p_2) + \cdots + p_n * (1 - p_n) + \lambda)}$$

$$O_{value} = \frac{\left((y_1 - p_1) + (y_2 - p_2) + \cdots + (y_n - p_n)\right)}{\sum_{i=1}^{n} [p_i * (1 - p_i)] + \lambda}$$

$$O_{value} = \frac{Sum\ of\ Residuals}{\sum [Previous\ Probability * (1 - Previous\ Probability)] + \lambda}$$

Similarity Score Equation Derive

$$Similarity\ Score = \frac{1}{2} \frac{(g_1 + g_2 + \cdots + g_n)^2}{(h_1 + h_2 + \cdots + h_n + \lambda)}$$

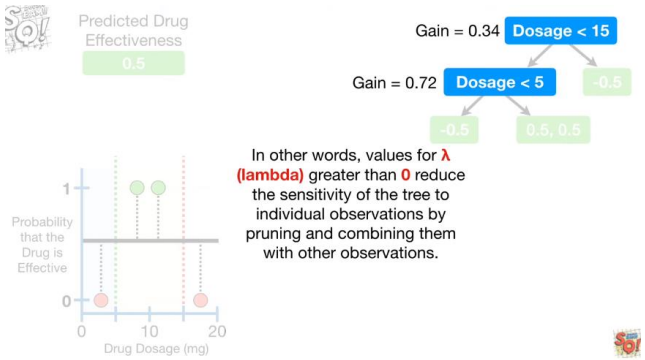
$$Similarity\ Score = \frac{(Sum\ of\ Residuals)^2}{\sum [Previous\ Probability * (1 - Previous\ Probability)] + \lambda} \rightarrow Omit \frac{1}{2}$$

Create Tree

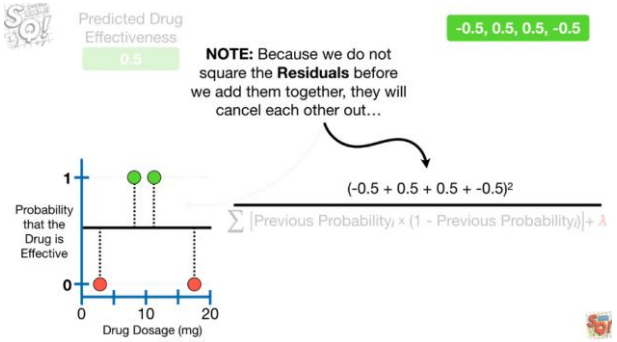
$$Similarity\ Score = \frac{(Sum\ of\ Residuals)^2}{\sum[p_i * (1 - p_i)] + \lambda}$$

$$Sum\ of\ Residual = Data\ Point - \mu$$

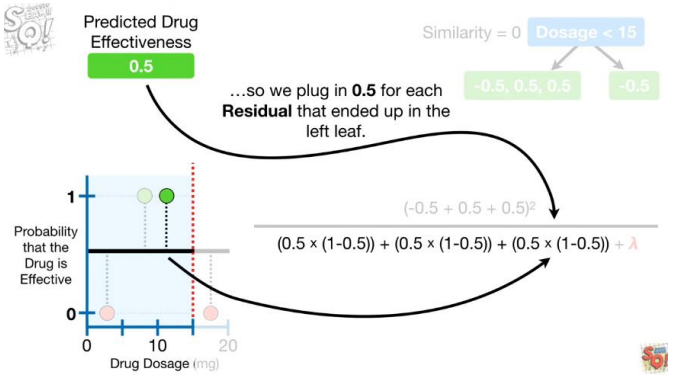
$$\lambda\ (lambda) \rightarrow Regularization\ Parameter$$



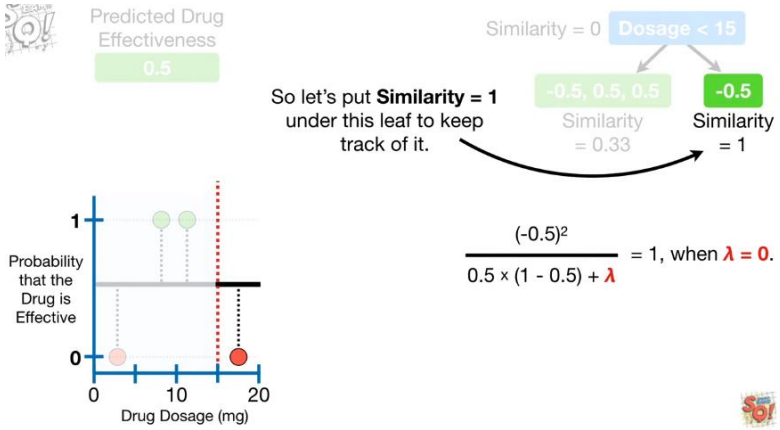
λ 的值越大，算出的 Similarity Score 就越小，而 Gain 也会随之而变小，在 Pruning 的时候，更容易把树 Prune 掉。



首先先使用所有 Data Point 与先前的初始值来计算出 Sum of Residuals 然后取平方。当得到了 Similarity Score 之后，就能继续创建这棵树。对两个 Data Point 之间的距离取平方，然后计算 Similarity Score。

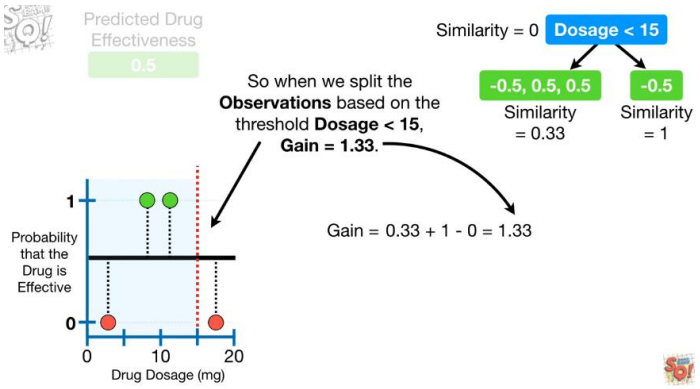


如上图，Threshold 设置在第三与第四的 Data Point 取平均得到$Dosage < 15$，但是也需要对第一第二， 第二第三的 Data Point 取平方后计算。Denominator 的部分，有多少个 Data Point 就有多少个 $p_i * (1 - p_i)$。

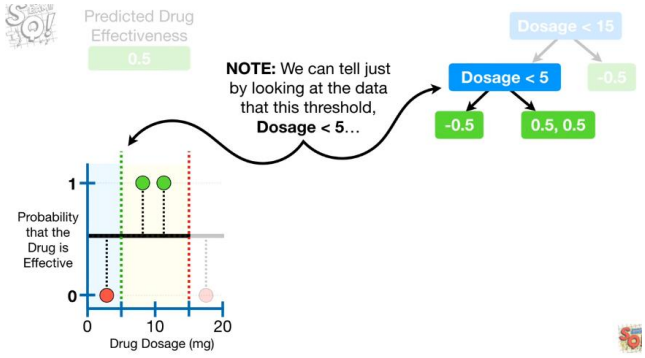


当计算完了左叶与右叶的 Similarity Score 之后，就需要计算 Gain。

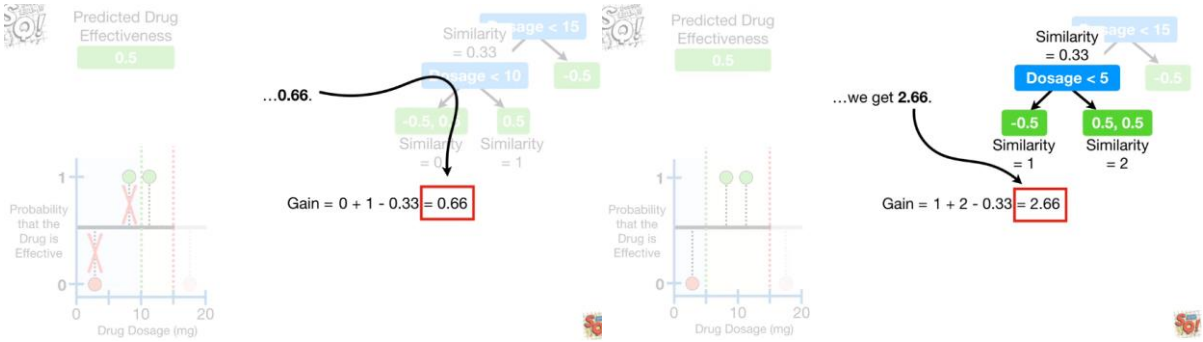
$$Gain = Left_{Similarity} + Right_{Similarity} - Root_{Similarity}$$



计算完 Gain 之后，选出有着最大的 Gain 的 Threshold 当作这棵树的 Branch。



之后继续使用同样的方法来选出下一层的 Threshold。

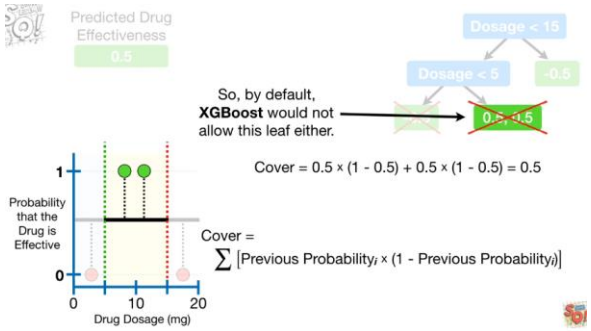


如上图，计算出了$Dosage < 5$ 与 $Dosage < 10$ 的 Gain 之后，$Dosage < 5$ 有着比较高的 Gain 所以选定的 Threshold 就是 $Dosage < 5$。只要 Leaf 里面有多过 1 个值，就能 Further Split，直到这棵树到达设定的 Level。

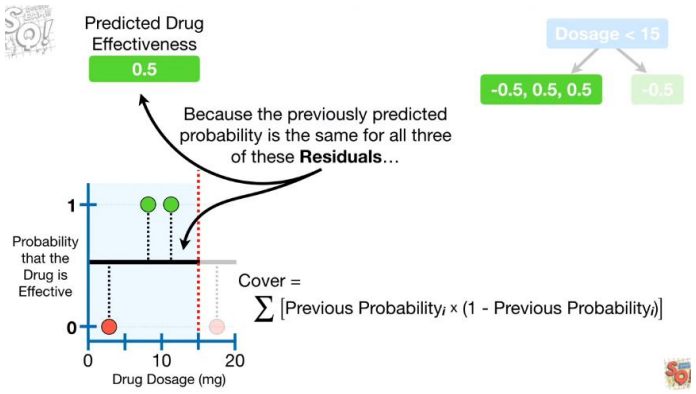
$$Cover = \sum[p_i * (1 - p_i)]$$

$$\text{Minimum Child Weight} \rightarrow \text{Cut Leaf When Cover} < \text{Minimum Child Weight}$$

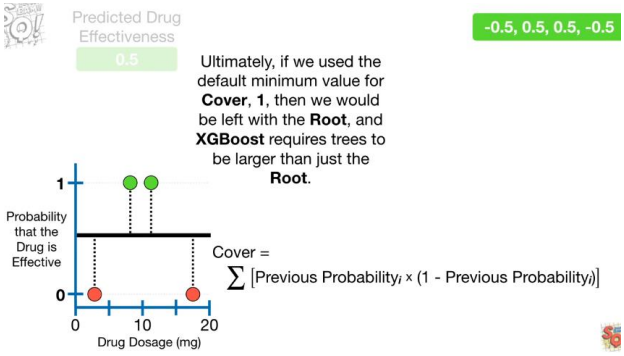
之后需要计算 Cover，Cover 在 XGBoost 里是用来裁剪 Leaves 的，Default Cover Value 是 1。当算出来的叶子的 Cover 小于 Minimum Child Weight，就代表这个叶子需要被裁剪。



如上图，$Dosage < 5$ 的两个叶子算出来的 Cover 都小于 1，所以这个 Threshold 也会被裁剪。



裁剪之后，就如上图，左叶里面有着 $Dosage < 5$ 两片叶子里的所有值。

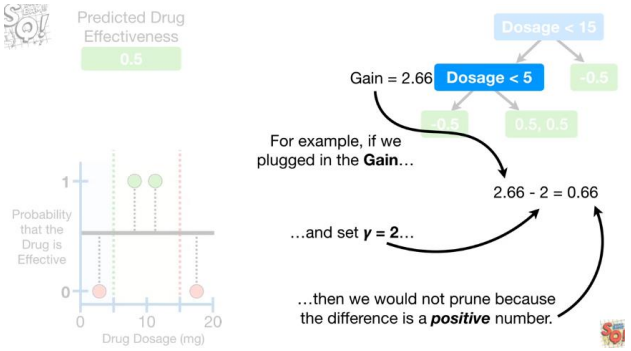


这个 Minimum Child Weight 需要通过测试，因为 XGBoost Requires Tree to be Larger Than Root，当 Minimum Child Weight 太大的话会被裁减成剩下 Root。

计算完 Cover 后，裁剪完之后，需要计算 Gain 与 γ 的差别后，来决定要不要 Prune 这个 Threshold。

$$Pruning\ Decision = Gain - \gamma \begin{cases} < 0, & Prune \\ \geq 0, & Dont\ Prune \end{cases}$$

$$\gamma \rightarrow 越大越容易Prune更多的Branch$$

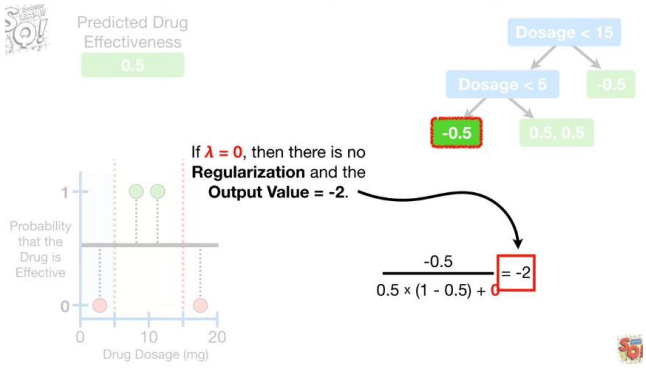


如上图，设置的 $\gamma = 2$，这时候计算出的 Pruning 值大于 0 所以该 Threshold 不会被 Prune 掉，叶子没被 Prune 掉，Root 也不会被 Prune 就算算出的 Pruning 值小于 0。

当 Pruning 好之后，就能计算这棵树的 Output 值，在 XGBoost for Classification 里，树的 Output 值是一个几率。

$$Output\ Value = \frac{Sum\ of\ Residuals}{\sum[p_i * (1 - p_i)] + \lambda}$$

$$\lambda \rightarrow Same\ Value\ as\ Similarity\ Score\ \lambda$$

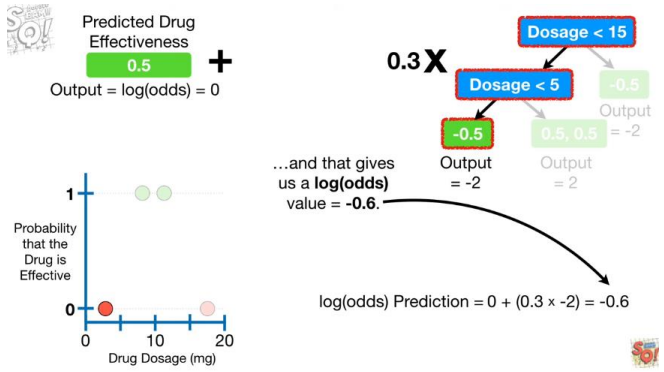


λ 的值越大，代表算出来的 Output Value 会越小。

当计算完所有叶子的 Output Value 之后，第一颗树也完成了。这时候就可以来做 Prediction。XGBoost 的 Prediction 与其他 Boosting 方法一样。当时在 XGBoost for Classification 里，首先需要把初始值转换成 $\log(odds)$ 。

$$odds = \frac{p}{1-p}$$

$$\log(odds) = \log\left(\frac{p}{1-p}\right)$$



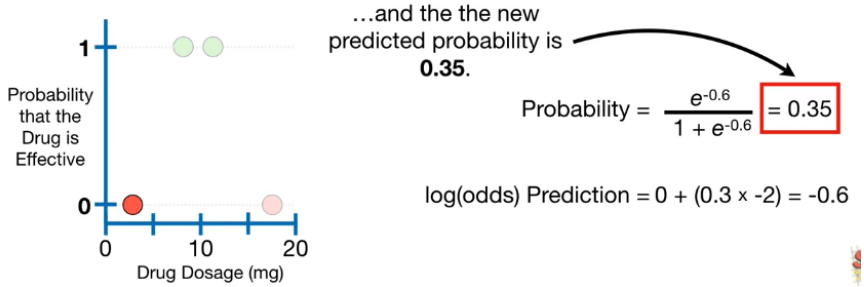
$$Total\log(odds)\ \ Prediction = \log(p_i) + \varepsilon * Current\ Tree\ Output$$

$$\varepsilon \rightarrow Learning\ Rate\ 0.3\ by\ Default$$

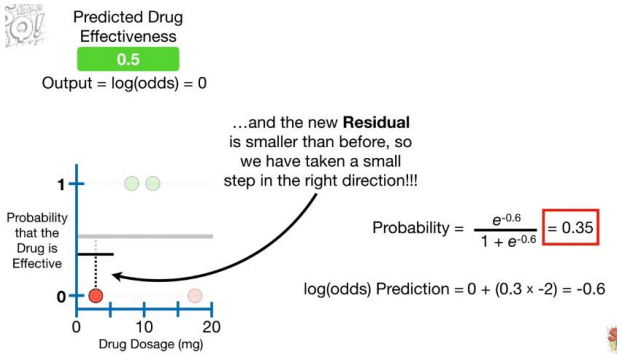
如上图，初始值求$\log(odds)$ 后再加上 Learning Rate 乘与第一颗树的 Output 值。而这个 Output 值的总和就是 $Total\ \log(odds)$。

得到了这个 $Total\ \log(odds)$ 之后，需要把这个值转换成几率 Probability。

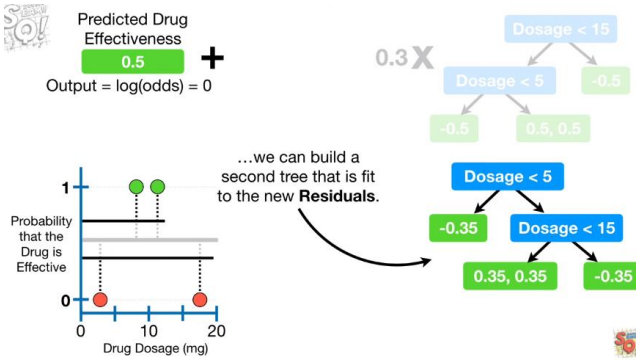
$$Probability = \frac{e^{\log(odds)}}{1 + e^{\log(odds)}}$$



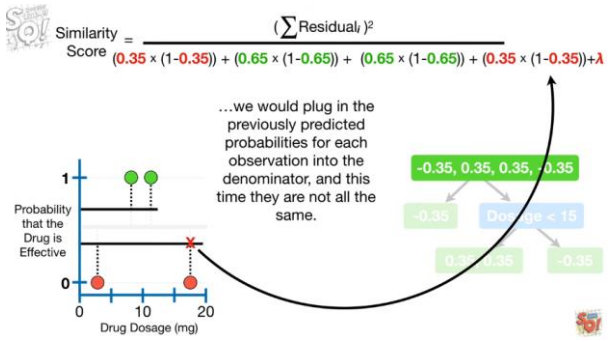
如上图，将计算出的 $Total\log(odds)\ Prediction \rightarrow -0.6$ 放入 Probability 的 Equation 得到了 0.35。



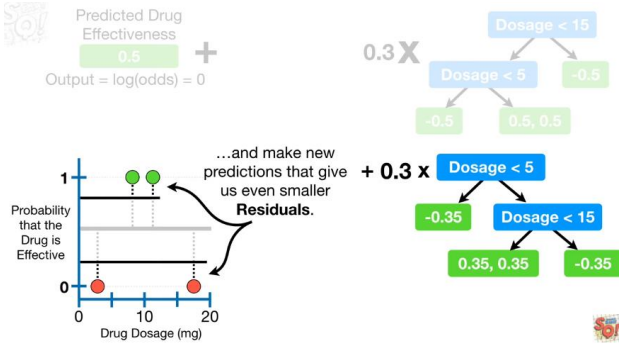
这个结果比原来只有初始值的 Prediction 进步了。



当对 n 笔 Data Point 都进行计算后，就能求出 n 笔新的 Residuals，就能继续创建下一棵树，直到创建完设定的树的数目，或者 Residual 已经非常小。



在创建下一棵树的时候，需要注意，这时候的 $p_i * (1 - p_i)$ 就是不一样了，需要使用这个 Data Point 与之间结果的 Residual 值。



如上图，使用了新的 Residual 来创建下一棵树，又能得到新的一组 Residuals 然后又能继续创建新的树。

XGBoost Optimization

Approximate Greedy Algorithm

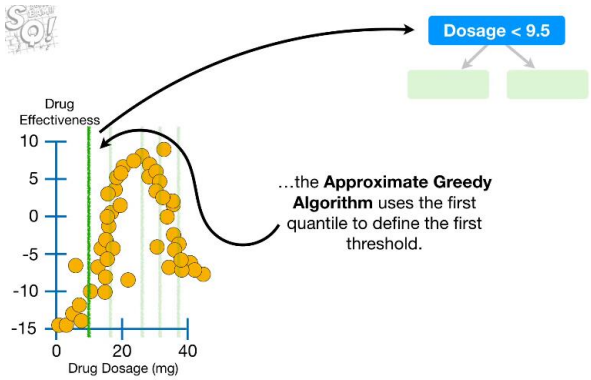
在 XGBoost 里，创建一棵树是将每 2 个 Data Point 取平均然后算出这个 Threshold 的 Gain，选出有着最大的 Gain 的 Threshold，这个就是典型的 Greedy Algorithm，因为不会考虑到之后的问题，只是选出当前最好的结果。

如果 XGBoost 不使用 Greedy Algorithm 来选择 Threshold 的话，就会等到整棵树都建立好后，在选择最好的树，每一颗树都有着不同的 Thresholds，这样做会让建立树的时间变长。可是当有着很大 Dataset 的时候，这个 Greedy Algorithm 的方法会有很大的计算量，导致变得很慢。

Dosage	Mass (kg)	Favorite Number	Other Stuff	Drug Effectiveness
10	63	32132	etc...	-7
34	72	12	etc...	-3
21	55	1001	etc...	7
etc...	etc...	etc...	etc...	etc...

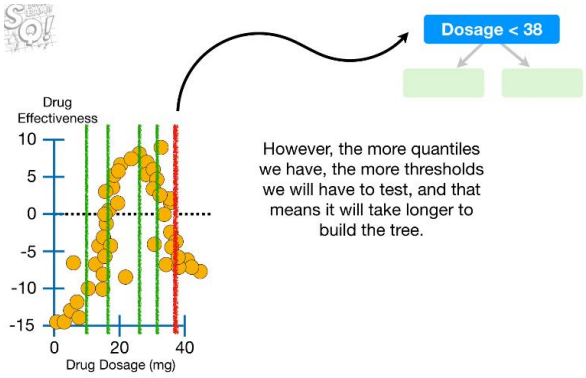
...then checking every single threshold in every single variable...
...would...
...take...
...forever.

当 Dataset 大又不止只有一个 Parameter 的时候，会让计算量很大，也可能算不出来。这时候需要用到 Approximate Greedy Algorithm 的方法。



Instead of 对每 2 个 Data Points 取平均来计算 Gain，这里的方法是将所有 Data Points 分成 Quantiles。只用 Quantiles 的值来当作 Threshold 然后计算 Gain。

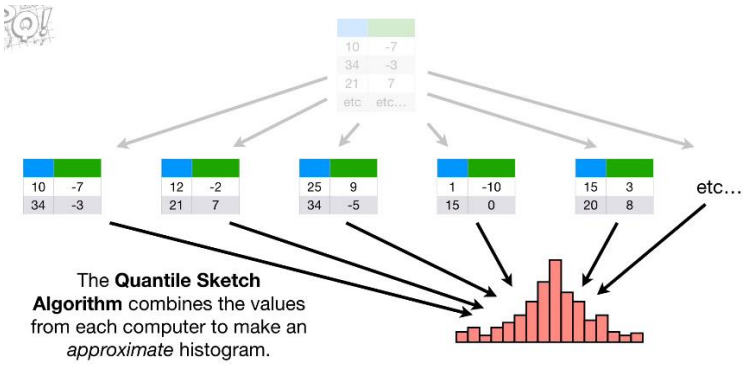
如果 Quantiles 的数目小，计算速度也会变很快，但是算出来的 Prediction 结果会很不好。



如上图，有着 5 个 Quantiles，与只有一个 Quantiles 的 Threshold 比起来，结果会好很多，但是计算量也会变大。在 XGBoost 里，By Default 会有大概 33 个 Quantiles。

Parallel Learning

当有着很大的 Dataset，不能一次过把这些 Data fit 进电脑的 Memory 的时候，要把 Dataset 里的所有 Data 按顺序 Sort 好和找出 Quantiles 的值会比较困难。这时候，需要使用 Sketches Algorithm 来解决这个问题，它可以很快的给出 Approximate Solutions。

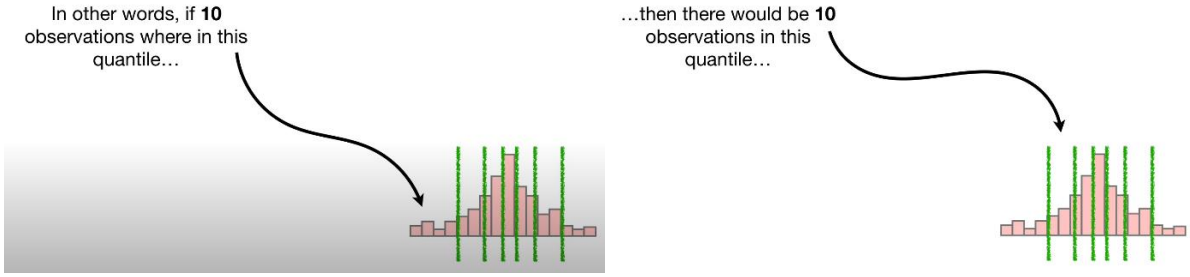


它的做法是将一个大 Dataset 分成好几份，然后使用不同的计算机来计算，计算完成后，将计算结果整合在一起能够得到一个 Approximate Histogram。

然后这个 Approximate Histogram 再来计算 Approximate Quantiles。

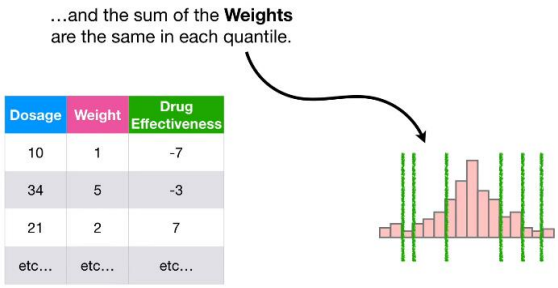
Weighted Quantile Sketch

上面解释的是 Normal Quantile Sketch，但是在 XGBoost 里面，使用的是 Weighted Quantile Sketch。计算出来的 Quantiles 是 Weighted 的。

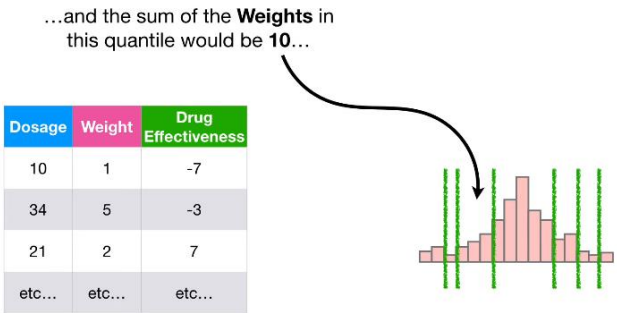


Weighted Quantiles 的意思就是每一个 Quantiles 里，都有着同样数目的 Observations。

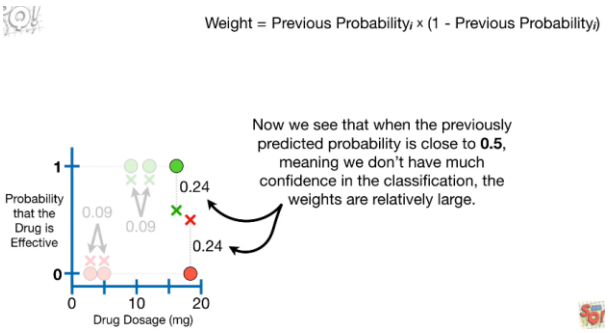
在 Weighted Quantiles 里，每一个 Observation 都有一个 Weight。



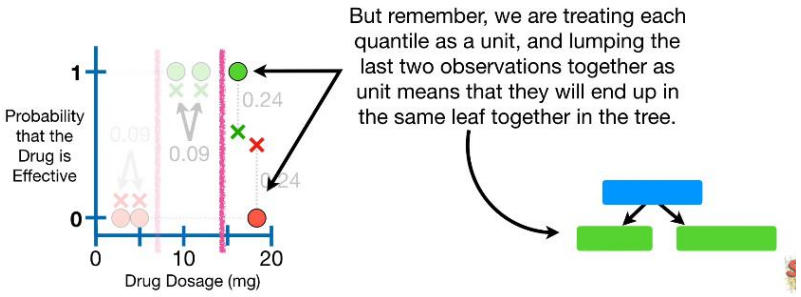
而这个 Sum of the Weight 在每一个 Quantile 里面是一样的。



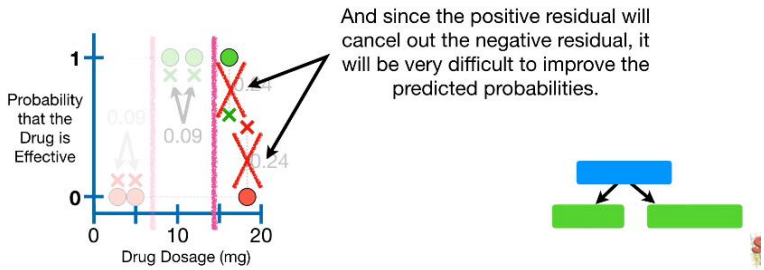
如上图，每一个 Quantile 里面都有着同样数目的 Weight。如果第一个 Quantile 里有 10 个 Weight，那么接下来的每一个 Quantile 里都有 10 个 Weight。这个 Weight 就是所谓的 Cover，也就是 Second Derivative of the Loss Function，也叫 Hessian。在 XGBoost for Regression 里，$Hessian = 1$。而在 XGBoost for Classification 里，$Hessian = p * (1 - p)$。



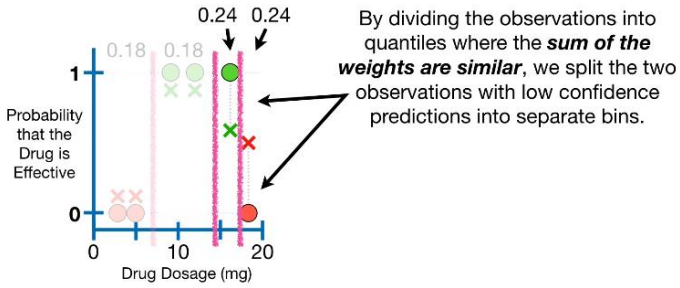
在 XGBoost for Regression 里，Weighted Quantile Sketch 是使用了 Previous Predicted Probability 来计算。当 Previous Predicted Probability 接近 0.5 的时候，算出来的 Weight 也会是比较大，也代表机器也不确定这个 Data 的答案。



这时候如果使用 Same Quantile，就是说第一个 Quantile 里有 2 个 Data Point，剩下的也需要有一样数目的 Data Points。

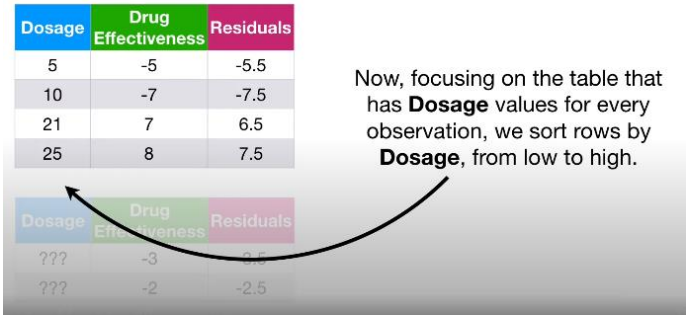


如上图，如果 Data Points 之间距离比较大，那么算出来的 Positive Residual 会把 Negative Residual 减去，这会导致预测值不能被进步。这时候就需要 Weighted Quantiles。

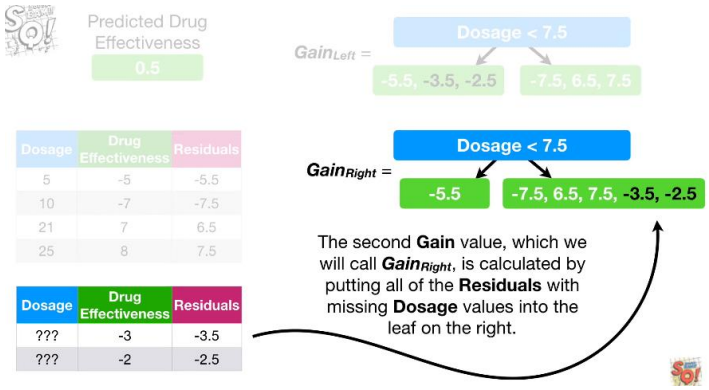


XGBoost 里每一个 Quantiles 里 Weight 的总和都是比较相似，而不是有着同样数目的 Data Points。如上图，0.18，0.18，到了第五个 Data Point 的时候，有着 0.24，所以这个 Data Points 自己就是一个 Quantile。这样的好处可以把有着 Low Confidence 的 Data 更加仔细的分类。

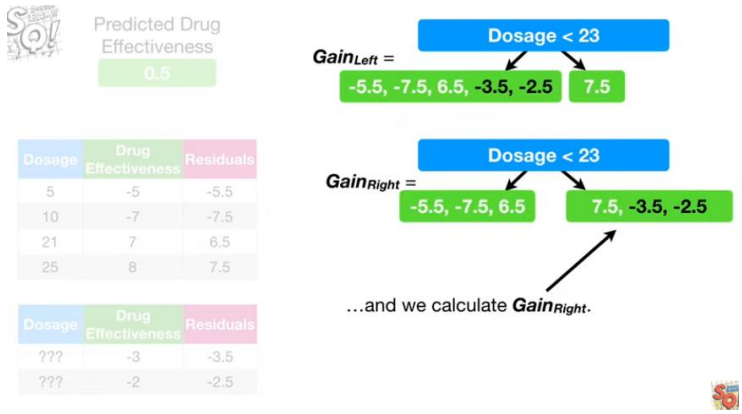
Sparsity-Aware Split Finding



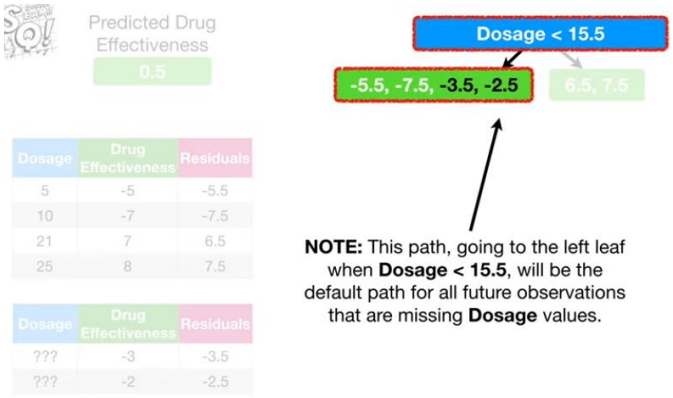
在 Dataset 里面，有着 Missing Value 的情况，把 Missing Value 的 Data 取出来集合在一起，然后将没有 Missing Value 的 Data 组合在一起按顺序排好。



首先先计算第一与第二笔 Data 作为 Threshold，然后就去的了上图的这颗树，这时候将全部 Missing Value 的 Data 的 Residuals 放入 Left Leaf 然后计算$Gain_{left}$。然后再将多有 Missing Value 的 Data 放入 Right Leaf 然后计算$Gain_{right}$。

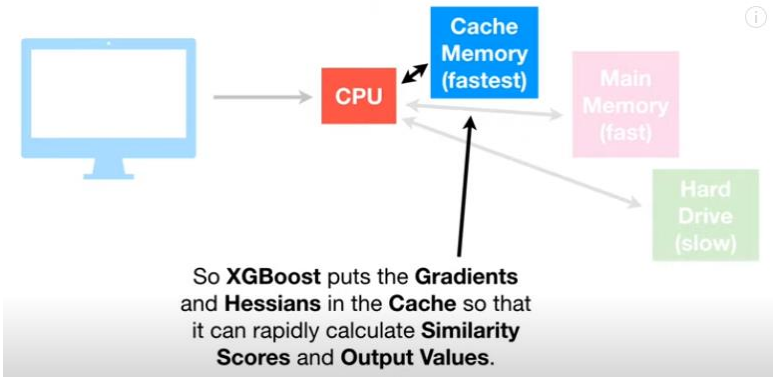


然后再继续计算第二笔与第三笔 Data 的平均作为 Threshold，再重复上面同样的步骤计算得到新的$Gain_{left}$ 和 $Gain_{right}$。一直重复直到最后第二与最后一个 Data 作为 Threshold 为止。然后选出有着最大 Gain 的 Threshold。



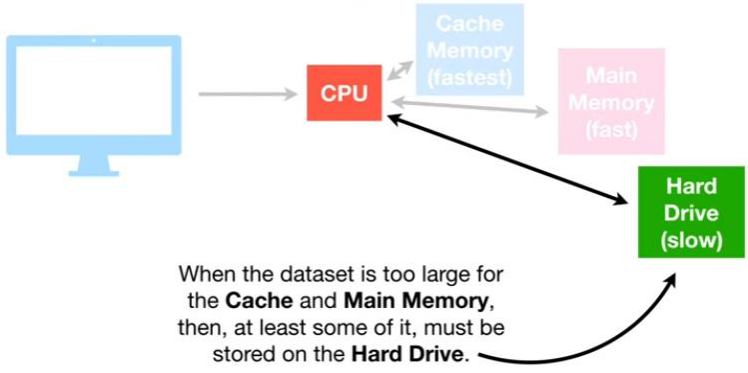
这时候选出的有着最大 Gain 的 Threshold 是 $Dosage < 15.5$，这个 Threshold 将用来预测所有 Missing Value。所有的 Missing Value 都会去到 $Dosage < 15.5$ 的 Path。

Cache-Aware Access



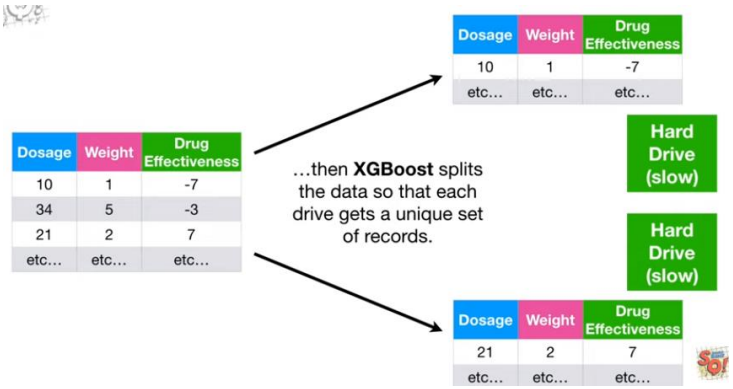
上图是一个电脑 CPU Access Memory 的简单架构，CPU Access Cache Memory 的速度是最快，然后到 Main Memory 再到 Hard Drive。在计算 XGBoost 的时候，XGBoost 把 Gradient 与 Hessian 的计算放在 Cache，这会让计算 Similarity Score 与 Output Values 的速度非常快。

Blocks for Out-of-Core Computation



当 Dataset 太大的时候，不能把所有 Data 都受灾 Cache 和 Main Memory 的时候，就需要用到 Hard Drive，但是 Hard Drive 的速度非常慢。XGBoost 为了让这个速度提升起来，会把 Data 都

压缩起来。虽然 CPU 在运用这些 Data 的时候需要做 Decompressing，但是还是比不压缩来的快很多。



当有不止一个 Hard Drive 的时候，XGBoost 会使用一个叫 Sharding 的方法来 Access 存储在 Hard Drive 里的 Data。XGBoost 会把 Dataset 里的 Data 分成 Unique Set，当需要读取的时候，能够同时读取两个 Hard Drive 里的 Data。