Suppose there are K trees (n_estimators = K). Then the prediction of the model for a given input x_i is

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i)$$

The loss of the model is calculated as:

$$L(\phi) = \sum_i l(\hat{y}_i, y) + \sum_k \Omega(f_k)$$

Where the first term is the loss of each data point, while the second term is a complexity penalty of each tree.

The parameters are f_k , so we want to minimize this by differentiating with respect to f_k and finding the optimal f_k . However this can't be done because the f_k are functions. Thus we train in an additive manner.

At boosting iteration t, we have

$$L^{(t)} = \sum_i l(y_i, \hat{y}_i{}^{(t-1)} + f_t(x_i)) + \Omega(f_t) \tag{1}$$

f_t is our candidate tree that we're thinking of adding to the ensemble.

At this point, we introduce some tricks to reduce this to a tractable algorithm. The first step is using 2nd order approximation, by Taylor's theorem.

Taylor: f differentiable at a, then Taylor expansion around a is

$$f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + ...$$

For compactness we'll write the first order differential, $f'(a)$ as g and the second order differential, $f''(a)$ as h.

Use Taylor for the loss function. Keep in mind that $\hat{y}$ plays the role of x in the theorem above, and that y is a constant here (it's a label, and we know its value).

Thus, we have that

$$\begin{aligned} l(y, \hat{y}) &\approx l(y,a) + g(\hat{y} - a) + \frac{1}{2}h(\hat{y} - a)^2 \\ &= l(y,a) + g(\hat{y}^{(t-1)} + f_t(x) - a) + \frac{1}{2}h(\hat{y}^{(t-1)} + f_t(x) - a)^2 \\ &= l(y, \hat{y}^{(t-1)}) + gf(x) + \frac{1}{2}hf_t^2(x) \end{aligned}$$

Where the last line follows by choosing $a = \hat{y}^{(t-1)}$.

We can substitute this into (1) and remove constant terms to see that the loss at boosting iteration t is

$$L^{(t)} = \sum_i [g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \Omega(f_t)$$

Since y, $\hat{y}^{(t-1)}$are already known, they are constants and can be removed. The sum is for all data points (i.e. each data point has its own g_i and h_i , and we don't forget the complexity of our candidate tree f_t. Keep in mind that the differentiation in g_i and h_i is with respect to the previous prediction $\hat{y}^{t-1}$. Our next step is to open up the complexity term Ω for further simplification.

Substitute

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

T : Number of leaves of the candidate tree.

λ : Regularization parameter for the leaf scores w_j in the tree.

γ : Regularization parameter for the number of leaves in the tree.

w_j : Leaf score of leaf j.

Note that in the list of hyperparameters for xgboost, γ is known as 'mininum split loss'. It isn't obvious from here why it has such a name yet, but as we progress further we shall see the reason behind its name.

With the above substitution, we have

$$\begin{aligned}
L^{(t)} &= \sum_i [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^T w_j^2 \\
&= \sum_i [g_i w_j(x_i) + \frac{1}{2} h_i w_j^2(x_i)] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^T w_j^2 \qquad (2) \\
&= \sum_{j=1}^T [(\sum_{i\in I_j} g_i) w_j + \frac{1}{2}(\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T \qquad (3) \\
&= \sum_{j=1}^T [G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2] + \gamma T \qquad (4)
\end{aligned}$$

We got to (2) by simply re-writing $f_t(x_i)$ in a more suggestive manner - each data point x_i would be sent down the tree to finally reach a leaf, with corresponding score w_j. $w_j(x_i)$ is the same meaning as, 'the leaf score w_j that we would get if we sent x_i down the tree'.

To move to (3), we move from summing over the data points to summing over the leaves. The first summand in (2) was for all data points, from $i = 1$ to $i = n$, while picking up each point's gradient g_i and hessian h_i in the sum. As in (2) , all data points x_i will eventually be sent to all leaves, so we can group together the data points falling in each leaf. Their output w_j is the same, though their individual gradients g_i and hessians h_i are possibly different.

Finally, to get to (4) we simply rewrite $G_j = \sum_{i\in I_j} g_i$ and $H_j = \sum_{i\in I_j} h_i$ for compactness.

Now that we have made this simplification, we can find the optimal leaf score w_j^* by differentiating $L^{(t)}$ with respect to w_j , setting equals zero and solving. Thus we have

$$\frac{\partial L^{(t)}}{\partial w_j} = G_j + (H_j + \lambda) w_j^* = 0 \implies w_j^* = -\frac{G_j}{H_j + \lambda}$$

Thus the prediction on each leaf is dependent on the gradients of the points in that leaf, the hessian of the points in that leaf, and the regularization parameter λ which is a hyperparameter option, known as 'reg_lambda'.

We can substitute this into the loss function $L^{(t)}$:

$$\begin{aligned}
L^{(t)} &= \sum_{j=1}^T [-\frac{G_j^2}{H_j + \lambda} + \frac{1}{2}\frac{G_j^2}{(H_j + \lambda)}] + \gamma T \\
&= \sum_{j=1}^T [-\frac{1}{2}\frac{G_j^2}{(H_j + \lambda)}] + \gamma T
\end{aligned}$$

This is now a scoring function for our tree structure - given any candidate tree, we immediately check this function and the one with the lowest output wins. But practically, we can't grow all trees and test them - instead, we will grow the tree layer by layer. Consider the case of one node, and the tree wondering if it should perform another split. Right before the split, the node is carrying a contribution to the loss function equal to

$$-\frac{1}{2}\frac{G_j^2}{(H_j + \lambda)} + \gamma$$

If it attempts a split, where there once was one leaf there are now two. The contribution to the loss function is

$$-\frac{1}{2}\frac{G_L^2}{(H_L + \lambda)} - \frac{1}{2}\frac{G_R^2}{(H_R + \lambda)} + 2\gamma$$

Where we've used the subscript L and R to denote the left leaf and right leaf respectively.

As such, the split should be attempted only if the contribution to the loss of the children split is less than the parent split. I.e. we want

$$\sum children < parent$$

And then we'll get a gain of

$$\begin{aligned} Gain &= parent - \sum children \\ &= -\frac{1}{2}\frac{(G_L+G_R)^2}{(H_L+H_R+\lambda)} + \gamma - [-\frac{1}{2}\frac{G_L^2}{(H_L+\lambda)} - \frac{1}{2}\frac{G_R^2}{(H_R+\lambda)} + 2\gamma] \\ &= \frac{1}{2}\{\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{(H_L+H_R+\lambda)}\} - \gamma \end{aligned}$$

With this, we can see why γ is known as minimum-split-loss. So the algorithm will consider splitting a leaf based on a feature by checking the above equation - and thus the tree is grown layer by layer.

Up until now, we have not specified just what G and H is. (Remember - the differentiation is against $\hat{y}^{t-1}$!). So we'll make that concrete by considering two cases - quadratic loss and logistic loss.

For convenience of writing, we'll set $\hat{y}^{t-1} = p$. One can think of p as 'prediction', since it's basically the prediction of the previous ensemble at $t-1$. It'll also become apparent when we consider logistic loss why we want p.

Quadratic Loss

The quadratic loss function is $l(y,p) = (y-p)^2$. Thus we have that $\frac{dl}{dp} = p - y$ and $\frac{d^2l}{dp^2} = 1$. In this case, min_child_weight is the number of data points in the leaf.

Logistic Loss

I can never remember the formula for logistic loss, so it helps to reconstruct it from first principles. Being logistic, we're thinking of classification, just like a Bernoulli variable:

$$y = \begin{cases} 1 & p \\ 0 & 1-p \end{cases}$$

where it can take the value 1 with probability p or 0 with probability $1-p$. Then the MLE estimate of p is $MLE(p) = \prod p^y(1-p)^{1-y}$. Every $y=1$ we observe contributes p to the likelihood, while every $y=0$ contributes $1-p$. From here, all we need to do is take the logarithm and append a negative, since we want to maximize MLE but the framework requires a loss to minimize.

$$l(p) = -\{\sum(ylnp + (1-y)ln(1-p)\}$$

With this loss, we can find the gradient and hessian:

$$\begin{aligned} \frac{dl}{dp} &= \sum(\frac{-y}{p} - \frac{(1-y)}{1-p}) \\ &= \sum\frac{-y(1-p)+(1-y)p}{p(1-p)} \\ &= \sum\frac{p-y}{p(1-p)} \end{aligned}$$

If we check the source code for xgboost, we'll find that is not what the gradient should be. Where did we go wrong? The answer is the previous prediction has been *sigmoided*, to forcefully put it between 0 and 1 as we required in classification. So we don't actually have p. We have $\sigma(p)$, where σ is the sigmoid function.

In the above, we replace every p we see with $\sigma(p)$ instead. Then we add in another $\frac{d\sigma}{dp} = \sigma(1-\sigma)$, by the chain rule. With this, we end up with

$$\begin{aligned} \frac{dl}{dp} &= \sum\frac{(\sigma(p)-y)}{\sigma(p)(1-\sigma(p))} * \frac{d\sigma}{dp} \\ &= \sum\frac{(\sigma(p)-y)}{\sigma(p)(1-\sigma(p))} * \sigma(p)(1-\sigma(p)) \\ &= \sum(\sigma(p)-y)) \end{aligned}$$

Which is exactly what is in the source code. The hessians are:

$$\frac{d^2l}{dp^2} = \sigma(1-\sigma)$$

And min_child_weight is the sum of this. What shall we make of this expression? It is close to zero when σ is close to 0 or 1, i.e. the model is very confident in its prediction. This is maximized when $\sigma = 0.5$, when the model is basically tossing a coin to decide. So we can think of this as the amount of 'unexplained mass' in a leaf. If the leaf is classifying very well, you need a lot of points in this leaf before the tree considers another split. If the leaf isn't, relatively less points are needed before the tree considers another split.