Classification with GBDTs

Alina Kochocki (kochocki@msu.edu)
(Dated: August 2021)

A. Intention of Document

This document is intended as an in-depth explanation of the algorithmic processes used to train gradient boosted decision trees (GBDTs), and their potential application in future IceCube event selections. General background on the topic of decision trees (applicable in regression and classification) is given, as this basic training process is employed in the more powerful method of gradient boosting. Discussion will be highly focused on GBDTs, and the approach to multi-class classification used in the software, XGBoost. Such a classifier is necessary in cascade analyses - distinguishing between cascade events, starting tracks and muon cosmic-ray background. A general procedure for model construction and special considerations for a Gen2 geometry are presented.

While this work has focused on building additional intuition around classification with GBDTs, much of the document has followed existing presentations. Specifically, the general XGBoost model documentation (https://xgboost.readthedocs.io/en/latest/tutorials/model.html), and discussion on classification from Hans Niederhausen (https://www.stonybrook.edu/commcms/grad-physics-astronomy/_theses/niederhausen-hans-may-2018.pdf) were referenced heavily.

B. Introduction to Decision Trees

Decision trees and other machine learning models provide optimizable solutions for non-parametric functions. The general problem involves a dataset, $D = \{(y_i, \mathbf{x}_i); 1 \leq i \leq N\}$, of N entries, each associated with a vector of features, $\mathbf{x}_i$, and known target variable, y_i. We would like to form a predictive model for the target variable, y_i, informed by its known associations with the observed features, $\mathbf{x}_i$. Ideally, the legitimate trends extracted from the training dataset, D, are then used to predict the unknown target variables of other data.

This model seeks to reduce the difference between predicted and known target values (residuals) by identifying similarly target-valued sub-populations through their observables, $\mathbf{x}_i$. The final prediction, f_l, assigned to the lth sub-population will have improved ability to represent homogeneous, associated entries. This residual value is commonly incorporated into a loss function, L, to be minimized. As all possible trees cannot be tested, the splitting process used to identify sub-populations is performed iteratively, always valuing splits which minimize our objective function. Training proceeds by a recursive binary decision process, favoring target variable homogeneity in newly identified sub-samples. In a simple approach:

1. The entire training dataset constitutes the root node. A first pass estimate of the target value may be produced from an average over all entries.

2. A first split in the dataset is made by scanning over the space of each observable feature distribution. The split feature and value is chosen to minimize the loss function, generating two new sub-populations of increased homogeneity. Reevaluating the target variale in

each case would provide two improved values, increasingly representative of their associated entries. Generally, the entry-wise differences between the target variable truth and new, tighter estimate will decrease.

3. The two nodes produced by each new split are again evaluated. Splits are generated until the target variables of each node are identical (common in the case of classification), or the node contains a single entry (regression). Terminal nodes are commonly referred to as leaves.

4. The resulting structure, f, represents the mapping or decision making process used to assign predicted values to data. As there is a set number of final predictions depending on the number of developed leaves, we can address these values either as a function of an individual data point (i.e. $f(x_i)$) or by indexing the leaf itself (the value f_l for the lth leaf). A new data entry is passed through the same decision process to assign a prediction.

This approach uses a single tree to generate predictions for new data. The accuracy of this quick approach can be improved upon by slowing the learning process. Later sections will discuss gradient boosted decision trees, which utilize the same training process to iteratively predict corrections to the previous tree's best estimate. Here, the result of a single tree is replaced by the learning ability of t trees.

C. Classification, Non-Parametric Likelihood Optimization

Decision trees are commonly used to classify data, able to yield a probability of association for multiple classes. The target values are bounded on $[0, 1]$. In the case of binary classification, y_i is the known probability of association of an ith entry with the first class, $(1 - y_i)$ with the second. The binary log-loss function is given as:

$$L_{\text{binary}} = -\frac{1}{N}\sum_{i}^{N}(y_i \log(\hat{p}_i) + (1 - y_i)\log(1 - \hat{p}_i)). \tag{1}$$

Here, $\hat{p}_i$ is the predicted value of the entry. If not associated with the first class ($y_i = 0$), the first term is eliminated, and the second is minimized for predicted values close to zero. The first term is similarly minimized for entries of the first class with predicted values near one.

Notably, the loss function, L, is the negative log-likelihood. Other BDT procedures may seek to optimize alternate statistics. However, the loss function can generally be related to the log-likelihood, providing an optimization method for the case of non-parametric functions.

As bounded target values are more difficult to work with, binary probabilities can be mapped to the space of real numbers with the logit function:

$$w_i = \text{logit}(\hat{p}_i) = \log\left(\frac{\hat{p}_i}{1 - \hat{p}_i}\right). \tag{2}$$

The sigmoid function transforms this value back to its probability representation:

$$\hat{p}_i = \sigma(w_i) = \frac{1}{1 + \exp(-w_i)}. \tag{3}$$

In the more general case of N_c classes, the multi-class log-loss function is used:

$$L_{\text{multi-class}} = -\frac{1}{N}\sum_{i}^{N}\sum_{j}^{N_c} y_{i,j}\log(\hat{p}_{i,j}) = \sum_{j}^{N_c}\Big(-\frac{1}{N}\sum_{i}^{N} y_{i,j}\log(\hat{p}_{i,j})\Big). \tag{4}$$

Each datum has N_c probabilities, corresponding to its predicted association with N_c classes. Always, $\sum_j^{N_c}\hat{p}_{i,j} = 1$. A similar transformation is used for ease in computation, expressing the predicted probability through the softmax function:

$$\hat{p}_{i,z} = \sigma_{softmax}(z, \mathbf{w}_i) = \frac{\exp(w_{i,z})}{\sum_j^{N_c}\exp(w_{i,j})}. \tag{5}$$

This can be substituted into the multi-class log-loss equation for an objective function in terms of unbounded variables, $w_{i,j}$:

$$L_{\text{multi-class}} = \sum_{j}^{N_c}\Big(-\frac{1}{N}\sum_{i}^{N} y_{i,j}\log\Big(\frac{\exp(w_{i,j})}{\sum_m^{N_c}\exp(w_{i,m})}\Big)\Big) \tag{6}$$

$$= \sum_{i}^{N}\frac{1}{N}\Big(-\Big(\sum_{j}^{N_c} y_{i,j}w_{i,j}\Big) + \log\Big(\sum_{j}^{N_c}\exp(w_{i,j})\Big)\Big). \tag{7}$$

Both generic decision trees and GBDTs will require an initial guess. We may take an average as a zeroth prediction for all class probabilities:

$$\begin{aligned}\hat{p}_1 &= \frac{\sum_i^N y_{i,1}}{N}\\ \hat{p}_2 &= \frac{\sum_i^N y_{i,2}}{N}\\ &\cdots\\ \hat{p}_{N_c} &= \frac{\sum_i^N y_{i,N_c}}{N}\end{aligned} \tag{8}$$

D. Model Complexity

Training a decision tree with only a loss function as the minimization objective generally leads to unwanted results. Where the ideal training performance means capturing only legitimate data trends expected to extend to similar populations, the unguided split-making process tends to overtrain. If only completing when the final result sub-populations (leaves) reach total homogeneity, rules will develop to exact some expected proportion of outlier and noise-like entries, unimportant features of the dataset, not representative of future sets used for prediction. Ideally, the branching process will only select those sub-populations with enough evidence for inclusion, ignoring spurious entries, and trends which are not statistically significant as to be described well by the training process. This is generally performed through trial and error, heuristic methods. There is not an accepted, best feature for this purpose, but a penalty term of known analytic form may be

introduced and combined with the loss function for minimization:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{l}^{T} \sum_{j}^{N_c} f_{l,j}^2. \tag{9}$$

Here, T is the number of leaves on a given tree, scaled by the user-set value of γ. $f_{l,j}$ of the second term refers to the output prediction of the lth leaf for the jth class. The user-set value, λ sets the importance of their squared magnitude in the penalty term. This later component will be of use in gradient boosted trees, where one might prefer smaller output values to increment previous predictions.

The result is the two-term objective function used by XGBoost:

$$\text{obj} = L + \Omega(f). \tag{10}$$

The training process is refined by the addition of $\Omega(f)$. In optimizing the objective function, a potential split is assessed both on its ability to minimize the loss function from an initial high, and to minimize model complexity from an initial low. Inhomogeneous nodes may now end as leaves, if the addition to model complexity of all potential splits outweighs reduction of the log-loss function.

E. Gradient Boosting, XGBoost Implementation

As suggested, the prediction gained from an ensemble of trees will provide an improved result over an individual. While certain concepts carry from the single tree case, training is performed with a different intention (or objective!). Gradient boosting is the process of generating sequential trees, each minimizing the errors of the previous. Instead of a single tree trained to predict our target, each tree is trained to predict a corrective value with respect to the previous tree's result. A final prediction can be found from summing over the corresponding leaf scores, an approximate summation over predicted quasi-residuals. This result is described as:

$$w_{i,j}^{(t)} = \sum_{k}^{t} f_j^{(k)}(\mathbf{x}_i), \tag{11}$$

where t is the number of trees, and $f_j^{(k)}(\mathbf{x}_i)$ is the output leaf score of the kth tree for the jth class. In the case of classification, the desired probability is found from a softmax transformation. Explicitly, the summation as a function of tree level:

$$\begin{aligned}
w_{i,j}^{(0)} &= 0 \\
w_{i,j}^{(1)} &= f_j^{(1)}(\mathbf{x}_i) = w_{i,j}^{(0)} + f_j^{(1)}(\mathbf{x}_i) \\
w_{i,j}^{(2)} &= f_j^{(1)}(\mathbf{x}_i) + f_j^{(2)}(\mathbf{x}_i) = w_{i,j}^{(1)} + f_j^{(2)}(\mathbf{x}_i) \\
&\ldots \\
w_{i,j}^{(t)} &= \sum_{k}^{t} f_j^{(k)}(\mathbf{x}_i) = w_{i,j}^{(t-1)} + f_j^{(t)}(\mathbf{x}_i).
\end{aligned} \tag{12}$$

So, the output score, $w_{i,j}^{(t)}$, is an incremental improvement over $w_{i,j}^{(t-1)}$, exacting the best final prediction as a summation. The output of each tree represents a predicted quasi-residual of the previous tree level's best target value prediction.

Ideally, new trees will be added to minimize the original objective function, where the summed best prediction is used as input:

$$\text{obj}^{(t)} = \sum_{i}^{N}\sum_{j}^{N_c} L(y_{i,j}, w_{i,j}^{(t)}) + \sum_{k}^{t}\Omega(f^{(k)}) \tag{13}$$

$$= \sum_{i}^{N}\sum_{j}^{N_c} L(y_{i,j}, w_{i,j}^{(t-1)} + f_{t,j}(\mathbf{x}_i)) + \sum_{k}^{t}\Omega(f^{(k)}) \tag{14}$$

This function will seek the target variable increment or output score ($f_j^{(t)}(\mathbf{x}_i)$) which best adjusts the previous prediction, minimizing the objective function. Written exactly in eq. 13, the new tree would train on the value $f_j^{(t)}(\mathbf{x}_i)$, where the objective function (ignoring the complexity penalty) is minimized such that:

$$\sigma_{softmax}(j, f^{(t)}(\mathbf{x}_i)) = y_{i,j} - \hat{p}_{i,j}^{(t-1)}. \tag{15}$$

Specifically, the 'best' result value assigned to each leaf is the exact residual with respect to the previous prediction. In this case, the next tree will stop at the root node, as the best prediction with respect to the current tree is now perfect. This is not the ideal behavior, allowing the tree to train too quickly with poor predictive ability. Instead of a single tree making a few decisions to dominate the prediction process, learning the best prediction slower with successive trees will better map the features of the data. In practice, the individual trees are stopped prematurely from complexity control, never predicting the exact residual. This allows us to build a model out of many shallow trees, making small adjustments, and slowly converging to our target value, as opposed to a deep, individual tree. Commonly, individual trees are referred to as 'weak learners', where a composite model is a 'strong learner'.

To support a variety of log-loss functions, XGBoost recasts eq. 14 as it's second order Taylor expansion. As opposed to training for the exact residual, we will use an increment, $f_j^{(t)}(\mathbf{x}_i)$, in the direction of an improved objective (informed by the results of the previous tree):

$$\text{obj}^{(t)} = \sum_{i}^{N}\sum_{j}^{N_c}\left(L(y_{i,j}, w_{i,j}^{(t-1)}) + g_{i,j}f_j^{(t)}(\mathbf{x}_i) + \frac{1}{2}h_i(f_j^{(t)}(\mathbf{x}_i))^2\right) + \sum_{k}^{t}\Omega(f_k) \tag{16}$$

Here, $g_{i,j}$ and $h_{i,j}$ are the first and second order loss function derivatives with respect to the previous tree's best prediction, $w_{i,j}^{(t-1)}$:

$$g_{i,j} = \partial_{w_{i,j}^{(t-1)}} L(y_{i,j}, w_{i,j}^{(t-1)}), \tag{17}$$

$$g_{i,j} = \partial^2_{w_{i,j}^{(t-1)}} L(y_{i,j}, w_{i,j}^{(t-1)}). \tag{18}$$

This quasi-residual $f_j^{(t)}(\mathbf{x}_i)$ will improve upon the target value estimate of the previous tree when added to its prediction, $w_{i,j}^{(t-1)}$. The previous loss function contribution is still present

as $L(y_{i,j}, w_{i,j}^{(t-1)})$, while the space of the loss function with respect to the previous prediction is modeled as a parabola about some non-zero minimum. Ignoring the contribution from complexity, a given parabola will have a minimum for $f_j^{(t)}(\mathbf{x}_i)$ such that:

$$L(y_{i,j}, w_{i,j}^{(t-1)}) + g_{i,j} f_j^{(t)}(\mathbf{x}_i) + \frac{1}{2} h_i (f_j^{(t)}(\mathbf{x}_i))^2 < L(y_{i,j}, w_{i,j}^{(t-1)}). \tag{19}$$

The sign of this $f_j^{(t)}(\mathbf{x}_i)$ depends on the first and second order derivatives, but in general will serve as the properly signed correction to walk our last prediction towards the correct value. The gradients will shrink in absolute value as the prediction improves and the residuals decrease, leading to tighter parabolic spaces, and smaller increment $f_j^{(t)}(\mathbf{x}_i)$ values. This is the notion of 'gradient'. Similar to gradient descent algorithms, we've used second order approximation to search for a local minimum.

Further, we can explicitly solve for the best $f_{t,l,j}$ of the tth tree, lth leaf, jth class, and corresponding minimized objective function. This is the general formula evaluated by the software, XGBoost, and is inclusive of complexity. Dropping the constant terms established by the previous tree, the variable portion of the tth tree's objective function:

$$\text{obj}^{(t)} \approx \sum_i^N \sum_j^{N_c} \left(g_{i,j} f_j^{(t)}(\mathbf{x}_i) + \frac{1}{2} h_i (f_j^{(t)}(\mathbf{x}_i))^2 \right) + \gamma T + \frac{1}{2} \lambda \sum_l^T \sum_j^{N_c} f_{l,j}^2 \tag{20}$$

$$= \sum_l^T \sum_j^{N_c} \left(\big(\sum_{n \in I_l} g_{n,j} \big) f_{l,j}^{(t)} + \frac{1}{2} \big(\sum_{n \in I_l} h_{n,j} + \lambda \big) (f_{l,j}^{(t)})^2 \right) + \gamma T. \tag{21}$$

Here, the set I_l is the set of indices for all entries associated with the lth leaf. These corresponding derivatives may be summed:

$$G_{l,j} = \sum_{n \in I_i} g_{n,j}, \tag{22}$$

$$H_{l,j} = \sum_{n \in I_i} h_{n,j}, \tag{23}$$

yielding the simplified objective function form:

$$\text{obj}^{(t)} = \sum_l^T \sum_j^{N_c} \left(G_{l,j} f_{l,j}^{(t)} + \frac{1}{2} (H_{l,j} + \lambda) (f_{l,j}^{(t)})^2 \right) + \gamma T. \tag{24}$$

Again, the parabolic form is present, now reflecting the impact of complexity control paramters, λ and γ. We can solve directly for the best $f_{l,j}^{(t)}$. Differentiating with respect to $f_{l,j}^{(t)}$:

$$f_{l,j,best}^{(t)} = -\frac{G_{l,j}}{H_{l,j} + \lambda}. \tag{25}$$

This value will minimize our objective function, substituting back in:

$$\text{obj}_{best} = -\frac{1}{2} \sum_l^T \sum_j^{N_c} \frac{G_{l,j}^2}{H_{l,j} + \lambda} + \gamma T. \tag{26}$$

This is the equation used for training in XGBoost. The original log-loss equation and previous results are represented within. To make the decision process more explicit, we can consider the updating process used to determine new splits, as derived from the above objective form. Here, we define *Gain*:

$$Gain = \frac{1}{2}\sum_{j}^{N_c}\left(\frac{G_{L,j}^2}{H_{L,j}+\lambda} + \frac{G_{R,j}^2}{H_{R,j}+\lambda} - \frac{(G_{L,j}+G_{R,j})^2}{H_{L,j}+H_{R,j}+\lambda}\right) - \gamma \tag{27}$$

This value is evaluated for each proposed split (feature and feature value). Splits are chosen to maximize *Gain*, equivalent to minimizing the objective function. The first two components represent the negative objective contributions from a proposed split (leading to two new nodes, left and right denoted by L and R subscripts). The complexity gain from the split is represented by the final component, λ. This is to be compared to the third term, the pre-split objective contribution. This function is used to determine the potential improvement in the objective function, by simply subtracting the negative of the two potential objective values (before and after the potential split). Canceling constant terms established by earlier trees and unrelated leaves are not shown.

In the specific case of multi-class optimization, we consider the derivative of the log-loss function with respect to the previously predicted, transformed probability, $w_{i,z}^{(t-1)}$:

$$\partial_{w_{i,z}^{(t-1)}} L_{\text{multi-class}} = \partial_{w_{i,z}^{(t-1)}} \sum_{i}^{N} \frac{1}{N}\left(-\left(\sum_{j}^{N_c} y_{i,j} w_{i,j}^{(t-1)}\right) + \log\left(\sum_{j}^{N_c} \exp(w_{i,j}^{(t-1)})\right)\right) \tag{28}$$

$$= y_{i,z} + \frac{\exp(w_{i,z})}{\sum_{j}^{N_c}\exp(w_{i,z}^{(t-1)})} \tag{29}$$

$$= \hat{p}_{i,z} - y_{i,z} = \begin{cases} \hat{p}_{i,z} & \text{if } y_{i,z} \equiv 0 \\ \hat{p}_{i,z} - 1 & \text{if } y_{i,z} \equiv 1. \end{cases} \tag{30}$$

The gradients themselves are the residual values expected per class. Further, we can differentiate again for the second order derivative:

$$\partial^2_{w_{i,z}^{(t-1)}} L_{\text{multi-class}} = \partial_{w_{i,z}^{(t-1)}} \frac{\exp(w_{i,j}^{(t-1)})}{\sum_{k}^{N_c}\exp(w_{i,k}^{(t-1)})} = \hat{p}_{i,z}^{(t-1)}(1-\hat{p}_{i,z}^{(t-1)}). \tag{31}$$

As the probability estimate approaches its truth value (1 or 0), the values of both partials decrease, tightening the parabolic space used to estimate the loss function.

Once the trained set of trees is complete, generating a prediction for a new datum, $\mathbf{x}_i$, is straightforward:

$$\hat{p}_{i,z} = \frac{\exp(\sum_{k}^{t} f_z^{(k)}(\mathbf{x}_i))}{\sum_{j}^{N_c}\exp(\sum_{k}^{t} f_j^{(k)}(\mathbf{x}_i))}. \tag{32}$$

F. Training Control

Beyond the tree structures determined in training, hyperparameters are another set of tunable parameters which impact training and prediction. There are two of interest in the scope of this work:

- M: The total number of trees used for prediction.
- α: The learning rate used to scale the impact of summed leaf results. For simplicity, this was assumed 1 in the previous section. Where $\alpha \in (0, 1]$:

$$w_{i,j}^{(t)} = \sum_{k=1}^{t} \alpha f_j^{(k)}(\mathbf{x}_i) = w_{i,j}^{(t-1)} + \alpha f_j^{(k)}(\mathbf{x}_i). \tag{33}$$

A new prediction is found as:

$$\hat{p}_{i,z} = \frac{\exp(\sum_{k=1}^{t} \alpha f_z^{(k)}(\mathbf{x}_i))}{\sum_{j}^{N_c} \exp(\sum_{k=1}^{t} \alpha f_j^{(k)}(\mathbf{x}_i))} \tag{34}$$

The impact of each increment is reduced, slowing the walk to the final prediction.