XGBoost Cannot Extrapolate

Aaron John Danielson, PhD
Department of Computer Science, University of Texas at Austin
aaron.danielson@austin.utexas.edu

Introduction

XGBoost and its cousins are some of the most popular machine learning methods in use today. Their widespread use is attributed to many factors, including their success in public competitions, the ease with which features of different types can be integrated, the lack of common problems such as multicollinearity, and intrinsic handling of missing values. Building a robust model with a gradient-boosted tree is accessible even to practitioners with limited knowledge of machine learning. Several papers document the success of these models on tabular data compared to alternative methodologies, particularly deep learning, which often struggles with structured data due to its high data and computational requirements [4, 5, 6]. While this ease of use has contributed to their popularity, it has also led to potential overuse in situations where their limitations, such as the inability to extrapolate, become apparent.

In this article, we will explore one of the most problematic aspects of XGBoost: it cannot extrapolate. This limitation arises from the structure of decision trees, which form the backbone of gradient boosting algorithms.

Background on XGBoost

XGBoost, introduced by Chen and Guestrin [1], is a scalable tree ensemble method based on gradient boosting. The algorithm constructs a model by adding decision trees with leaf values that minimize a regularized loss function at each step. The following section explains the objective function, residual computation, leaf node values, and the optimization process in detail.

Model Structure

Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$ be a training dataset with n examples of features $x_i \in \mathbb{R}^m$ and labels $y_i \in \mathbb{R}$. XGBoost models the output as an additive ensemble of T

regression trees

$$\hat{y}_i = \sum_{t=1}^{T} f_t(x_i).$$

Each function f_t in the ensemble is defined by a tree structure $q_t : \mathbb{R}^m \to \{1, \ldots, K_t\}$ that assigns input points to one of the K_t leaf nodes and corresponding values $w_{q_t(\cdot)} \in \mathbb{R}^{K_t}$ such that

$$f_t(x_i) = w_{q_t(x_i)}.$$

Training the model amounts to learning the tree structure and associated leaf node values for all functions in the ensemble.

Training

The objective function combines a training loss and a regularization term

$$\mathcal{L}(\phi) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{t=1}^{T} \Omega(f_t),$$

where $l(y_i, \hat{y}_i)$ measures the difference between predicted and target values and $\Omega(f_t)$, given by

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

penalizes model complexity with γ controlling the penalty on the number of leaves K_t, and λ penalizing the size of the leaf values w_j.

XGBoost uses an additive approach to optimize the objective function. At each iteration t, the model adds a new tree f_t to update the fitted value to

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i).$$

The objective at step t,

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t),$$

can be approximated with a second-order Taylor expansion,

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)\right] + \Omega(f_t),$$

where the gradient g_i measures the slope of the loss function at $\hat{y}_i^{(t-1)}$,

$$g_i = \frac{\partial l(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)}},$$

and the Hessian measures the curvature of the loss function,

$$h_i = \frac{\partial^2 l(y_i, \hat{y}_i^{(t-1)})}{\partial(\hat{y}_i^{(t-1)})^2}.$$

Given a tree structure $q_t(\cdot)$, the objective function can be rewritten in terms of the leaf nodes as

$$\tilde{\mathcal{L}}^{(t)} = \sum_{k=1}^{K_t} \left[\left(\sum_{i \in S_k} g_i \right) w_k + \frac{1}{2} \left(\sum_{k=1}^{K_t} h_i + \lambda \right) w_k^2 \right] + \gamma K_t.$$

Minimizing the loss with respect to the weights obtain the solution

$$w_k^* = -\frac{\sum_{i \in S_k} g_i}{\sum_{i \in S_j} h_i + \lambda},$$

where S_k is the set of training instances assigned to leaf k. Then, if $x_i \in$ leaf k,

$$f_t(x_i) = w_k^*$$

such that the prediction for any input x_i at step t is constant within the region of feature space corresponding to leaf k.

The optimal weights learned during each iteration depend on the choice of tree structure. This is determined by evaluating potential splits to maximize the reduction in loss. For a split dividing the training instances S into S_L and S_R, the loss reduction is given by

$$\Delta\mathcal{L}_{\text{split}} = \frac{1}{2} \left[\frac{\left(\sum_{i \in S_L} g_i\right)^2}{\sum_{i \in S_L} h_i + \lambda} + \frac{\left(\sum_{i \in S_R} g_i\right)^2}{\sum_{i \in S_R} h_i + \lambda} - \frac{\left(\sum_{i \in S} g_i\right)^2}{\sum_{i \in S} h_i + \lambda} \right] - \gamma.$$

This formula quantifies the reduction in loss due to a split by evaluating the difference between the sum of squared gradients before and after the split, penalized by the regularization parameter γ. Given the tree structures $q_1(\cdot), \ldots, q_T(\cdot)$ and associated leaf node values $w^*_{q_1(\cdot)}, \ldots, w^*_{q_T(\cdot)}$, the final model is

$$\hat{y}(x) = \sum_{t=1}^{T} w^*_{q_t(x)}.$$

Partitioning of the Feature Space

XGBoost constructs an ensemble of decision trees, each of which partitions the feature space into disjoint regions corresponding to leaf nodes. For ease of notation, references to the particular tree number t are suspended temporarily. The splits along a feature dimension m result in a partition of $\mathbb{R}$ into disjoint intervals. The partitioning of each feature dimension can be placed into one of three general groups.

No Split: If no split occurs along dimension mm, then the entire range $(-\infty, \infty)$ forms a single interval.

Single Split: If there is a single split value s, then the dimension divides into a left interval $I_{\text{left}} = (-\infty, s)$ and right interval $I_{\text{right}} = [s, \infty)$.

Multiple Splits: A set of h intervals of the form $(-\infty, s_1)$, $[s_1, s_2)$, …, $[s_{h-1}, \infty)$, where $\{s_1, \ldots, s_{h-1}\}$ are the split points determined during training.

Let r_m be the number of intervals partitioning dimension m. The splits induce a set of $r = r_1 * \cdots * r_M$ hyperrectangles of the form

$$\mathcal{R}(j_1, \ldots, j_M) = I_{j_1} \times \cdots \times I_{j_M} \subset \mathbb{R}^M$$

where j_m indicates the interval index for interval m. Hence

$$\mathbb{R}^M = \bigcup_{j_1} \cdots \bigcup_{j_M} \mathcal{R}(j_1, \ldots, j_M).$$

Each hyperrectangle in the partition $\mathcal{R}(j_1, \ldots, j_M)$ corresponds to a unique leaf node in the tree $q(\cdot)$ such that every point x in this region receives the same prediction leaf node value equal to $w^*_{q(x)}$. This is the basis of XGBoost's inability to extrapolate: regions beyond the training data boundaries are assigned constant predictions based on the nearest partition.

The set of partitions induced by the ensemble of trees produces a finer partition of the feature space. Let

$$\mathcal{P}^{(t)} = \left\{ \mathcal{R}_j^{(t)} \mid j = 1, \ldots, r^{(t)} \right\}$$

denote the partition of $\mathbb{R}^M$ induced by the t-th tree where $\mathcal{R}_j^{(t)}$ is the j-th region in the partition and $r^{(t)}$ is the total number of subsets in the partition. Then the intersection of all T partitions is the finest-grained partition induced by ensemble defined as

$$\mathcal{P}_{\text{ensemble}} = \bigcup_{j_1=1}^{r_1} \cdots \bigcup_{j_T=1}^{r_T} \left(\bigcap_{t=1}^{T} \mathcal{R}_{j_t}^{(t)} \right).$$

Each region in $\mathcal{P}_{\text{ensemble}}$ is the intersection of one region from each tree. The prediction for a point $x \in \mathbb{R}^M$ is computed as the sum of the leaf values from all trees, weighted by the indicator function for the region to which x belongs

$$\hat{y}(x) = \sum_{\mathcal{R} \in \mathcal{P}_{\text{ensemble}}} \left(\sum_{t=1}^{T} w^*_{q_t(x)} \right) \mathbb{I}\left\{ x \in \mathcal{R} \right\}.$$

Hence the model can predict at most $|\mathcal{P}_{\text{ensemble}}|$ distinct values.

Consider a simple example in which training data is observed within the unit square $[0, 1] \times [0, 1]$. Suppose two decision trees are fit to this data, each splitting the feature space along specific cutpoints. The first tree partitions the feature space into regions based on a single cutpoint along feature 1 at $x_1 = 0.3$ and a single cutpoint along feature 2 at $x_2 = 0.5$. The second tree further partitions the space with a cutpoint at $x_1 = 0.5$ for feature 1 and a cutpoint at $x_2 = 0.9$ for feature 2.

The resulting partition of the feature space induced by the ensemble is the intersection of the regions formed by each tree. This produces a finer-grained partition, where each region corresponds to the intersection of one region from each tree. Figure 1 visually demonstrates how XGBoost partitions the feature space into disjoint regions through iterative splits along feature dimensions. Each region is represented by a unique color and labeled with its corresponding region index $\mathcal{R}_i$. This visualization demonstrates the key property of ensemble models like XGBoost: the ability to iteratively refine the feature space into increasingly precise regions. Note, however, that all regions except $\mathcal{R}_6$ are unbounded. This fact highlights a major limitation: regions outside the training data inherit constant predictions from the partition to which they belong, emphasizing the model's inability to extrapolate.

Impact on Extrapolation

This partitioning structure explains why XGBoost cannot extrapolate beyond the boundary of the training data. Any point outside the convex hull of the training data belong to exactly one set in the partition. Since no additional splits occur for unseen regions, the predictions cannot adjust dynamically based on new input values, limiting the model's extrapolation capability. To illustrate this point, consider the following example.

Suppose the training data is sampled uniformly from the range $[0, 1]$ with outputs following $y = x + \epsilon$, where $\epsilon \sim N(0, \sigma^2)$. Test data is sampled uniformly from the extended range $[-1, 2]$, with ϵ drawn from the same distribution as for the training data.

Figure 2 illustrates the predictions made on *inliers* (points within $[0, 1]$) and *outliers* (points outside $[0, 1]$) for varying levels of noise in the outputs. The figure highlights XGBoost's inability to extrapolate beyond the range of the training data. For all noise levels, the function learned by XGBoost remains constant beyond the boundaries of the training range. Specifically, for noise-free data ($\sigma^2 = 0$), XGBoost predicts $\hat{y}(x) = 0$ for all $x < 0$ and $\hat{y}(x) = 1$ for all $x > 1$. As noise increases ($\sigma^2 > 0$), predictions for *inliers* become less precise, but predictions for *outliers* remain constant. This limitation persists in classification problems as well. Consider a scenario where the true relationship

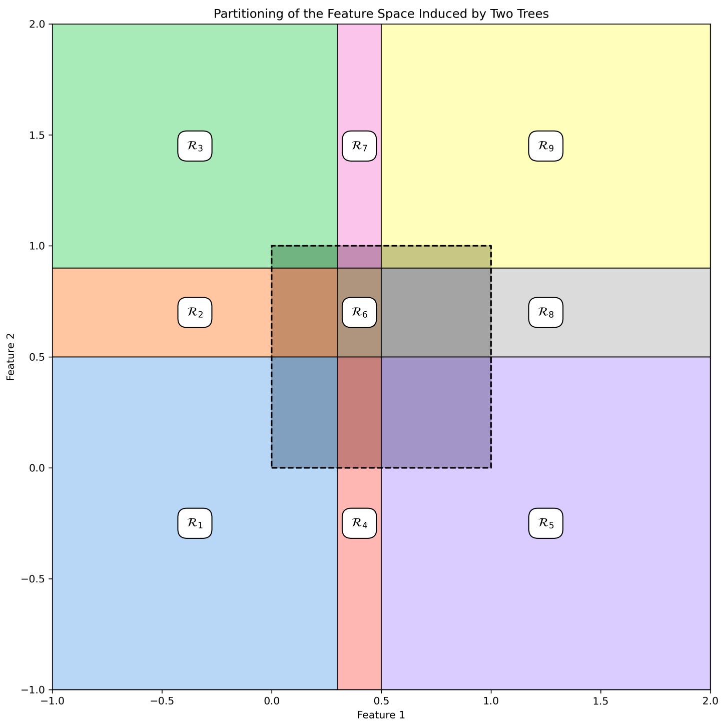

Figure 1: Illustration of the partitioning of feature space induced by the ensemble of two decision trees. The unit square is emphasized with a dashed border, and each region is uniquely labeled to highlight the finer-grained partition resulting from the intersection of tree partitions.

is given by

$$y \sim \text{Binomial}(\text{sigmoid}(x + \epsilon)),$$

where $\epsilon \sim \mathcal{N}(0, \sigma^2)$. The model is trained on inputs sampled uniformly from $[0, 1]$ and tested on inputs sampled from the extended range $[-1, 2]$. Figure 3 illustrates that predictions outside the training range $[0, 1]$ remain constant, reflecting the values of the nearest region in the partition induced by the trees. To highlight this bias, Figure 4 overlays the histograms of true and predicted probabilities in the outlier regions $[-1, 0)$ and $(1, 2]$ for the model trained with $\sigma^2 = 0$. This comparison demonstrates the systematic underestimation or overestimation of probabilities in these regions, depending on the proximity to training data boundaries. The next section coalesces these observations into a few formal propositions which help characterize the odd behavior of tree-based ensembles beyond the boundary of the training data.

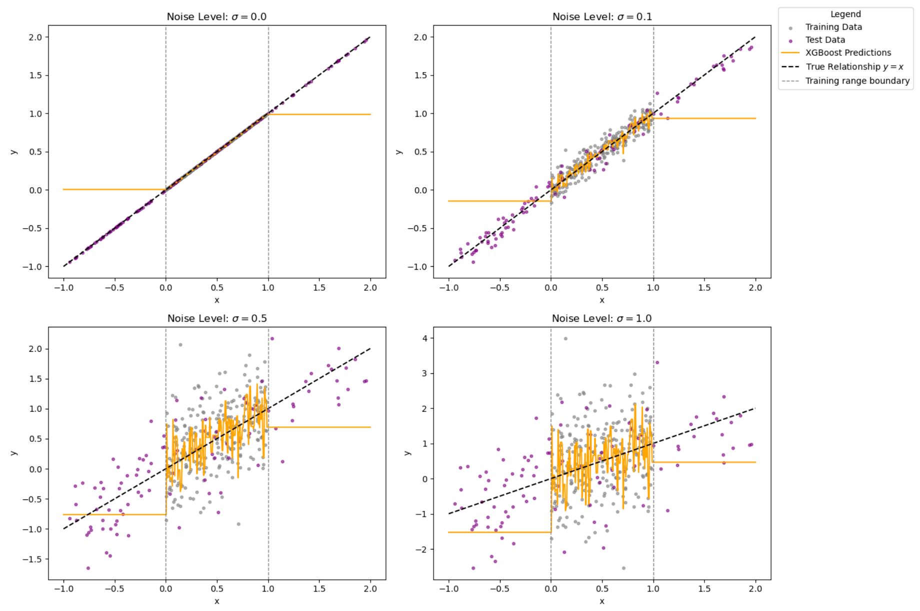

Figure 2: Predicted vs actual values for inputs in $[-1, 2]$. The learned function remains constant outside the training range, illustrating XGBoost's inability to extrapolate.

Formal Result

The prior sections characterize XGBoost as a method to partition the feature space and make constant predictions within each element in the partition. Since the training data is finite, subsets in the partition have infinite measure. Intuitively, points belonging to these regions return the same prediction no matter how far they are from the nearest training point. The following proposition makes this claim more precise.

Proposition 1. *Let $x \in \mathbb{R}^M$ be a boundary point of the convex hull of the training data $\mathcal{X}_{train}$. Then for all $\epsilon > 0$, there exists a coordinate $m \in \{1, \ldots, M\}$ in the feature space such that*

$$\hat{y}(x) = \hat{y}(x + \epsilon e_m),$$

or

$$\hat{y}(x) = \hat{y}(x - \epsilon e_m),$$

where e_m is the m-th standard basis vector.

Proof. Let $\mathcal{R}_x$ be the element of the partition $\mathcal{P}_{\text{ensemble}}$ induced by the ensemble of trees which contains x. Since x is a boundary point of the convex hull of $\mathcal{X}_{\text{train}}$, for all $\delta > 0$, $N_\delta(x) \not\subset \mathcal{X}_{\text{train}}$.

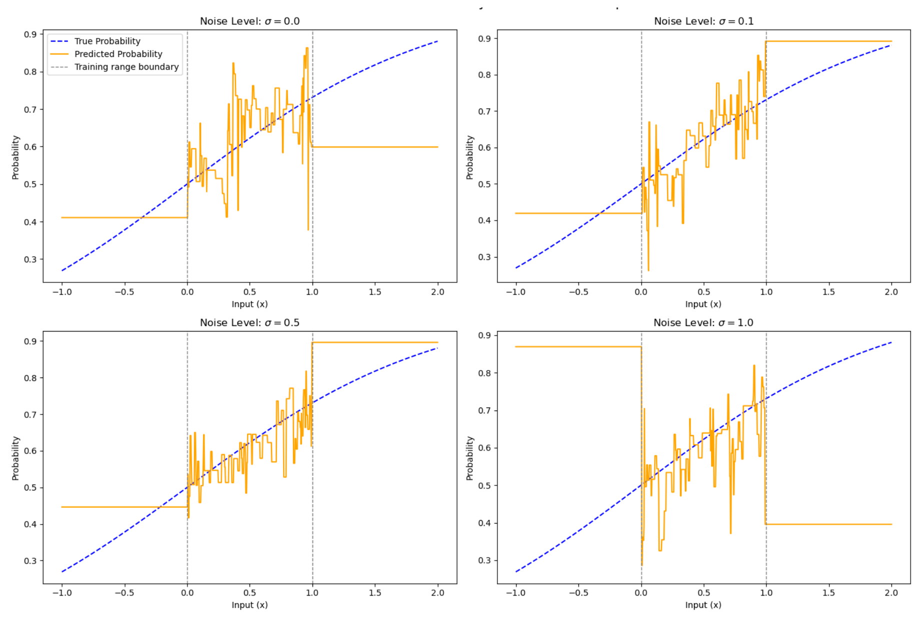

Figure 3: Predicted vs actual probabilities for inputs in $[-1, 2]$. Outside the training range, predictions are constant due to the nearest regions in the partition.

For at least one feature $m \in \{1, \dots, M\}$, the value x_m must lie in an unbounded interval of the form (s_l, ∞) or $(-\infty, s_u)$ where $s_l \in \mathbb{R} \cup \{-\infty\}$ and $s_u \in \mathbb{R} \cup \{\infty\}$. Without loss of generality, suppose $x_m \in (s_l, \infty)$. For any $\epsilon > 0$, $x_m + \epsilon \in (s_l, \infty)$. Thus, the perturbed point $\tilde{x} = x + \epsilon e_m$ remains in the same partition element $\mathcal{R}_x$.

Since $\tilde{x}$ and x belong to the same partition element, they are assigned to the same leaf nodes across all trees in the ensemble. Consequently, the prediction for $\tilde{x}$ is identical to that for x:

$$\hat{y}(x) = \hat{y}(\tilde{x}).$$

A similar argument holds if $x_m \in (-\infty, s_u)$, showing that $\hat{y}(x)$ is constant for perturbations along any unbounded dimension. This proves the proposition. □

This result formalizes the intuition that XGBoost partitions the feature space into fixed regions, preventing dynamic adaptation to new input values. Predictions are determined solely by the regions defined during training, with no mechanism for modeling trends beyond the training range. Unlike many real-world functions, the prediction does not change as $\epsilon \to \infty$. Note that for any arbitrary vector $v \in \mathbb{R}^M$, this claim does not hold. The point may enter a new

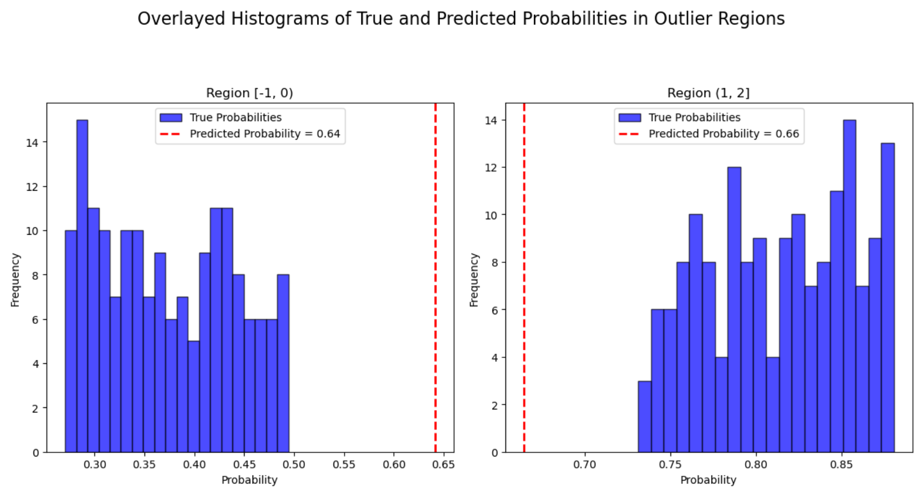

Figure 4: Overlayed histograms of true and predicted probabilities in $[-1, 0)$ and $(1, 2]$ for the noiseless model.

element of the partition such that the prediction changes.

Recent advancements in regression modeling have sought to address the extrapolation limitations of tree-based models. Gradient Boosted Spline Regression (GBSR) is one such method that combines the flexibility of gradient boosting with the smoothness of spline functions, enabling better handling of continuous trends in the data. Unlike XGBoost, which makes constant predictions within each partition, GBSR introduces smooth transitions across the feature space, potentially improving extrapolation performance in tasks requiring generalization beyond the training range. This hybrid approach demonstrates the potential of incorporating non-tree-based components into ensemble methods to overcome the rigid partitioning limitations of traditional gradient boosting algorithms [7].

In summary, XGBoost's inability to extrapolate arises directly from the partitioning structure of decision trees. While this limitation is acceptable in many applications, it poses challenges in tasks requiring extrapolation or generalization beyond the training range. Addressing this limitation may require integrating non-tree-based components or hybrid approaches.

References

[1] Chen, T., Guestrin, C. (2016). XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD Interna-*

tional Conference on Knowledge Discovery and Data Mining, 785?794. https://doi.org/10.1145/2939672.2939785

[2] Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5), 1189?1232. https://doi.org/10.1214/aos/1013203451

[3] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... Liu, T. Y. (2017). LightGBM: A highly efficient gradient boosting decision tree. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 3149?3157. https://proceedings.neurips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf

[4] Shwartz-Ziv, R., Armon, A. (2022). Tabular data: Deep learning is not all you need. *Information Fusion*, 81, 84?90. https://doi.org/10.1016/j.inffus.2021.11.011

[5] Grinsztajn, L., Oyallon, E., Varoquaux, G. (2022). Why do tree-based models still outperform deep learning on tabular data? *arXiv preprint arXiv:2207.08815*. https://arxiv.org/abs/2207.08815

[6] Borisov, V., Leemann, T., Seer, K., Haug, J. E., Pawelczyk, M., Kasneci, G. (2022). Deep neural networks and tabular data: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 1?21. https://doi.org/10.1109/TNNLS.2022.3163764

[7] Su, G., Yang, D., Zhu, J. (2018). Boosted Spline Regression Models for High-Dimensional Data. *Journal of the American Statistical Association*, 113(521), 108?121. https://doi.org/10.1080/01621459.2016.1268097