PKBoost: Mathematical Foundations

Shannon Entropy-Guided Gradient Boosting with Newton-Raphson Optimization

Pushp Kharat

Abstract

This document presents the complete mathematical framework underlying PKBoost, a gradient boosting algorithm that combines Newton-Raphson optimization with Shannon entropy-based information theory for improved performance on imbalanced classification tasks.

1 Introduction

PKBoost extends traditional gradient boosting by incorporating Shannon entropy as an information-theoretic guidance mechanism for tree splitting decisions, while maintaining Newton-Raphson optimization for parameter updates. This hybrid approach provides superior handling of imbalanced datasets.

2 Core Loss Function

2.1 Logistic Loss with Class Weighting

For binary classification, we use the weighted logistic loss:

Definition 1 (Weighted Log Loss)**.** *Given predictions $\hat{y}_i$ and true labels $y_i \in \{0,1\}$, the loss for sample i is:*

$$\mathcal{L}(y_i, \hat{y}_i) = -\left[w_i y_i \log(\sigma(\hat{y}_i)) + (1 - y_i)\log(1 - \sigma(\hat{y}_i))\right] \tag{1}$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function and w_i is the sample weight.

For imbalanced datasets with positive class ratio $r = \frac{\sum y_i}{n}$, we define:

$$w_i = \begin{cases} \sqrt{w_{\text{pos}}} & \text{if } y_i = 1 \\ 1 & \text{if } y_i = 0 \end{cases} \tag{2}$$

where $w_{\text{pos}} = \frac{1-r}{r}$ (capped at 8.0 to prevent instability).

3 Newton-Raphson Gradient Boosting

3.1 First and Second Order Derivatives

The gradient (first derivative) and Hessian (second derivative) guide the Newton-Raphson optimization:

Theorem 1 (Gradient and Hessian for Weighted Logistic Loss)**.** *For sample i with raw prediction f_i (in log-odds space):*

$$g_i = \frac{\partial \mathcal{L}}{\partial f_i} = w_i(\sigma(f_i) - y_i) \tag{3}$$

$$h_i = \frac{\partial^2 \mathcal{L}}{\partial f_i^2} = w_i \sigma(f_i)(1 - \sigma(f_i)) \tag{4}$$

where w_i applies the class weighting.

Proof. Starting with $\mathcal{L}(y_i, f_i) = -[w_i y_i \log(\sigma(f_i)) + (1 - y_i)\log(1 - \sigma(f_i))]$:

For the gradient:

$$\begin{aligned} g_i &= -\left[w_i y_i \frac{1}{\sigma(f_i)} \sigma'(f_i) - (1 - y_i) \frac{1}{1 - \sigma(f_i)} \sigma'(f_i) \right] \\ &= -\left[\frac{w_i y_i}{\sigma(f_i)} - \frac{1 - y_i}{1 - \sigma(f_i)} \right] \sigma(f_i)(1 - \sigma(f_i)) \\ &= w_i(\sigma(f_i) - y_i) \end{aligned}$$

For the Hessian, we differentiate the gradient:

$$\begin{aligned} h_i &= \frac{\partial g_i}{\partial f_i} = w_i \frac{\partial \sigma(f_i)}{\partial f_i} \\ &= w_i \sigma(f_i)(1 - \sigma(f_i)) \end{aligned}$$

□

3.2 Leaf Weight Optimization

Each leaf j in a decision tree receives an optimal weight computed via Newton's method:

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda} \quad (5)$$

where I_j is the set of samples in leaf j, and λ is the L2 regularization parameter.

3.3 Newton-Raphson Update

The model prediction is updated iteratively:

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \eta \cdot T_m(\mathbf{x}) \quad (6)$$

where η is the learning rate and T_m is the m-th tree with leaf weights computed using Equation 5.

4 Shannon Entropy Integration

4.1 Binary Shannon Entropy

Definition 2 (Binary Shannon Entropy)**.** *For a node with n_0 samples of class 0 and n_1 samples of class 1:*

$$H(p) = -p_0 \log_2(p_0) - p_1 \log_2(p_1) \quad (7)$$

where $p_0 = \frac{n_0}{n_0+n_1}$ and $p_1 = \frac{n_1}{n_0+n_1}$.

Maximum entropy occurs at $p_0 = p_1 = 0.5$ with $H = 1.0$ bit. Pure nodes have $H = 0$.

4.2 Information Gain

For a split dividing parent node P into left child L and right child R:

Definition 3 (Information Gain)**.**

$$IG = H(P) - \left[\frac{|L|}{|P|} H(L) + \frac{|R|}{|P|} H(R) \right] \quad (8)$$

Higher information gain indicates a more informative split.

4.3 Hybrid Split Criterion

PKBoost combines Newton-Raphson gain with information gain:

Theorem 2 (PKBoost Split Gain)**.** *The total gain for a split is:*

$$Gain_{total} = Gain_{Newton} + \alpha(d) \cdot IG - \gamma \tag{9}$$

where:

- $Gain_{Newton}$ *is the standard XGBoost gain (Equation 10)*
- $\alpha(d)$ *is a depth-dependent weight for entropy*
- IG *is the information gain (Equation 8)*
- γ *is the complexity penalty*

4.3.1 Newton Gain Component

The Newton-Raphson gain for a split is:

$$\text{Gain}_{\text{Newton}} = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma \tag{10}$$

where:

$$G_L = \sum_{i \in I_L} g_i, \quad H_L = \sum_{i \in I_L} h_i \tag{11}$$

$$G_R = \sum_{i \in I_R} g_i, \quad H_R = \sum_{i \in I_R} h_i \tag{12}$$

4.3.2 Adaptive Entropy Weighting

The entropy weight adapts based on node purity and tree depth:

$$\alpha(d, H_P) = \begin{cases} w_{\text{MI}} \cdot e^{-0.1d} & \text{if } H_P > 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

where w_{MI} is the mutual information weight hyperparameter and d is the current depth.
Rationale:

- High-entropy nodes benefit more from information-theoretic guidance
- Deeper nodes rely more on gradient information as patterns become more refined
- Pure nodes ($H_P \leq 0.5$) use gradient-only optimization

5 Regularization Framework

5.1 L2 Regularization

Leaf weights are regularized to prevent overfitting:

$$\Omega(T) = \lambda \sum_{j=1}^{J} w_j^2 \tag{14}$$

This appears in the denominator of Equation 5.

5.2 Tree Complexity Penalty

Each split incurs a complexity cost γ:

$$\text{Regularized Objective} = \mathcal{L} + \gamma \cdot (\text{number of leaves}) \tag{15}$$

5.3 Minimum Child Weight

A split is only valid if both children satisfy:

$$\sum_{i \in I_{\text{child}}} h_i \geq \text{min\_child\_weight} \tag{16}$$

This prevents splits creating leaves with insufficient support.

6 Histogram-Based Optimization

6.1 Feature Binning

Continuous features are discretized into K bins using quantile-based binning with adaptive allocation:

$$\text{Quantile}(q) = \begin{cases} \text{Linear}(q, 0, 0.1) & q \in [0, 0.25] \\ \text{Linear}(q, 0.1, 0.9) & q \in [0.25, 0.75] \\ \text{Linear}(q, 0.9, 1.0) & q \in [0.75, 1.0] \end{cases} \tag{17}$$

This allocates more bins to the tails of the distribution.

6.2 Histogram Aggregation

For each bin b and feature k, we maintain:

$$\text{Hist}_k[b] = \left(\sum_{i: x_i^{(k)} \in b} g_i, \sum_{i: x_i^{(k)} \in b} h_i, \sum_{i: x_i^{(k)} \in b} y_i, |b| \right) \tag{18}$$

6.3 Histogram Subtraction

For a parent node P with children L and R:

Lemma 1 (Histogram Subtraction Trick)**.**

$$Hist_R = Hist_P - Hist_L \tag{19}$$

This reduces computation by building histograms only for the smaller child.

7 Adversarial Vulnerability Detection

7.1 Vulnerability Score

For each prediction, we compute a vulnerability metric:

$$V_i = c_i \cdot e_i \cdot w(y_i) \tag{20}$$

where:

$$c_i = 2|\sigma(f_i) - 0.5| \quad \text{(confidence)} \tag{21}$$

$$e_i = |y_i - \sigma(f_i)| \quad \text{(error)} \tag{22}$$

$$w(y_i) = \begin{cases} \min\left(\frac{\sqrt{w_{\text{pos}}}}{100}, 5\right) & y_i = 1 \\ 1 & y_i = 0 \end{cases} \tag{23}$$

7.2 Exponential Moving Average

Vulnerability is tracked over time using EMA:

$$V_{\text{EMA}}^{(t)} = \beta V_i + (1-\beta) V_{\text{EMA}}^{(t-1)} \tag{24}$$

with $\beta = 0.1$ for slow decay preserving history.

8 Adaptive Metamorphosis

8.1 State Transition Thresholds

The model transitions between states based on calibrated thresholds:

$$\tau_{\text{alert}} = \mathbb{E}[V] \cdot \kappa_1(r) \tag{25}$$

$$\tau_{\text{meta}} = \mathbb{E}[V] \cdot \kappa_2(r) \tag{26}$$

where κ_1, κ_2 depend on class imbalance ratio r:

$$\kappa_1(r), \kappa_2(r) = \begin{cases} (1.5, 2.0) & r < 0.02 \\ (1.8, 2.5) & r < 0.10 \\ (2.0, 3.0) & r < 0.20 \\ (2.5, 3.5) & \text{otherwise} \end{cases} \tag{27}$$

8.2 Feature Utility Decay

Each feature maintains a utility score that decays exponentially:

$$u_j^{(t)} = \begin{cases} 1.0 & \text{if feature used at time } t \\ (1-\delta)^{\Delta t} u_j^{(t-\Delta t)} & \text{otherwise} \end{cases} \tag{28}$$

with decay rate $\delta = 0.0005$. Features with $u_j < 0.15$ are considered "dead".

8.3 Tree Pruning

Trees are pruned based on dependency on dead features:

$$D_m = \frac{\text{splits on dead features in } T_m}{\text{total splits in } T_m} \tag{29}$$

Trees with $D_m > 0.8$ are removed during metamorphosis.

Algorithm 1 PKBoost Training with Shannon Entropy

Input: Training data $(\mathbf{X}, \mathbf{y})$, max trees M, learning rate η
Initialize: $F_0(\mathbf{x}) = 0$ for all $\mathbf{x}$
Build histogram bins for all features
for $m = 1$ to M **do**
    Compute g_i, h_i for all samples using Equations 3, 4
    Sample features and instances (stochastic boosting)
    Build histogram Hist for sampled data
    Initialize tree T_m with root node
    Queue $\leftarrow$ [(root, Hist, depth=0)]
    while Queue not empty **do**
        Pop (node, hist, depth) from Queue
        if stopping criterion met **then**
            Set node as leaf with weight from Eq. 5
            continue
        end if
        Find best split using Eq. 9
        Partition samples into I_L, I_R
        Build Hist_L for smaller child
        Compute $\text{Hist}_R = \text{Hist} - \text{Hist}_L$
        Create left and right child nodes
        Add (left, Hist_L, depth+1) to Queue
        Add (right, Hist_R, depth+1) to Queue
    end while
    Update $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \eta T_m(\mathbf{x})$
    if early stopping criterion met **then**
        break
    end if
end for
Return: Ensemble $\{T_1, \ldots, T_M\}$

9 Complete Training Algorithm

10 Complexity Analysis

10.1 Time Complexity

- **Per tree:** $O(nd \cdot K \cdot \log n)$ where:
  - n = number of samples
  - d = number of features
  - K = number of bins (typically 32)
  - $\log n$ = tree depth
- **Histogram building:** $O(nd)$ using vectorized operations
- **Split finding:** $O(dK)$ per node
- **Total for M trees:** $O(Mnd \cdot K \cdot \log n)$

10.2 Space Complexity

- **Histograms:** $O(dK)$ per node
- **Transposed data:** $O(nd)$ stored once
- **Tree storage:** $O(M \cdot 2^{\text{depth}})$

11 Convergence Properties

Theorem 3 (Convergence of PKBoost)**.** *Under Lipschitz continuity of the loss function and bounded gradients, PKBoost converges to a local minimum with learning rate $\eta \in (0, 1]$:*

$$\lim_{M \to \infty} \mathcal{L}(F_M) = \mathcal{L}^* \tag{30}$$

where $\mathcal{L}^*$ is a local minimum.

The Shannon entropy term provides additional gradient information that can help escape shallow local minima, particularly in imbalanced scenarios.

12 Practical Considerations

12.1 Hyperparameter Selection

For imbalanced data with positive ratio r:

$$\text{Learning rate:} \quad \eta = 0.05 \cdot \begin{cases} 0.85 & r < 0.02 \\ 0.90 & r < 0.10 \\ 0.95 & r < 0.20 \\ 1.0 & \text{otherwise} \end{cases} \tag{31}$$

$$\text{Tree depth:} \quad d_{\max} = \lfloor \log(d) \rfloor + 3 - \mathbb{I}(r < 0.02) \tag{32}$$

$$\text{MI weight:} \quad w_{\text{MI}} = 0.3 \cdot e^{-\ln(r)} \tag{33}$$

12.2 Early Stopping

Training stops if validation metric (PR-AUC) fails to improve for E consecutive evaluations:

$$E = \min\left(\max\left(\frac{100}{\eta}, 30\right), 150\right) \tag{34}$$

Smoothing over last 3 evaluations reduces noise:

$$\text{Metric}_{\text{smooth}} = \frac{1}{3}\sum_{i=t-2}^{t} \text{Metric}_i \tag{35}$$

13 Conclusion

PKBoost unifies Newton-Raphson optimization with Shannon entropy-based information theory to create a robust gradient boosting framework particularly suited for imbalanced classification tasks. The adaptive metamorphosis mechanism enables online learning with drift detection and model self-healing capabilities.