Machine Learning Hierarchy and Implementation Strategy for Nuclear Level Matching

Level 1: The Foundation

Decision Tree

- **Concept:** A non-parametric supervised machine learning method that predicts target values by learning simple decision rules inferred from data features (Classification and Regression Tree, or CART).
- **Characteristics:** Low bias, high variance. Deep trees are unstable (sensitive to small data changes) and prone to overfitting (memorizing noise).
- **Role:** The "Weak Learner" (Base Estimator) in ensemble methods.

Level 2: The Strategy (Ensemble Learning)

Goal: Reduce variance (Bagging) or bias (Boosting) by combining multiple weak learners.

Strategy A: Bagging (Bootstrap Aggregating)

- **Logic:** Parallel execution to reduce variance.
- **Mechanism:** Generates M datasets via random sampling with replacement (bootstrap). Trains M independent trees. Final prediction is the average (regression) or majority vote (classification).
- **Key Algorithm:** Random Forest (Breiman, 2001).
- **Physics Limitation:** Averaging fails for "Hard Vetoes." If 1 tree detects a fatal Spin Mismatch (Probability approximately 0) but 99 trees see an Energy Match (Probability approximately 1), the average remains high (approximately 0.99). Physics requires the single veto to drive probability to 0.

Strategy B: Boosting

- **Logic:** Sequential execution to reduce bias.
- **Mechanism:** Iterative improvement. Tree m is trained to minimize errors (loss) of ensemble F_(m-1).
- **Strength for Physics:** Mimics a "Veto" system. If Tree 1 predicts a match, Tree 2 can detect a specific violation (e.g., Parity Mismatch) and output a large negative correction, effectively suppressing match probability.

Level 3: The Algorithm

Mathematical frameworks for implementing Boosting.

Algorithm A: AdaBoost (Adaptive Boosting)

- **Mechanism:** Sample Reweighting. At step m, it increases the weights of misclassified observations.
- **Reference:** Freund and Schapire (1997).
- **Verdict:** Reject. Sensitive to noisy data both theoretically and empirically. In nuclear data, experimental outliers (large errors) receive exponential weight, causing the model to fixate on anomalies rather than general trends.

Algorithm B: GBM (Gradient Boosting Machine)

- **Mechanism:** Functional Gradient Descent. At step m, a new tree is trained to predict the negative gradient (pseudo-residuals) of the loss function. It fits the error, not the data.
- **Reference:** Friedman (2001).
- **Verdict:** Superior. Optimizing differentiable loss functions (e.g., Log-Loss) makes it more robust to outliers than the exponential loss used in AdaBoost.

Level 4: The Package (Software Libraries)

Major libraries implementing Gradient Boosting.

Package	NaN Handling	Growth Strategy	Best Data Scale	Weakness for Physics	Verdict
Scikit-learn GradientBoosting (Legacy)	Fails (Crashes)	Level-wise	Small	Requires imputation (bias risk); slow; lacks regularization.	Reject
LightGBM (Microsoft, 2017)	Native (Safe)	Leaf-wise	Huge (>100k)	"Greedy" growth overfits small data; creates unbalanced trees.	Reject
Scikit-learn HistGradientBoosting (2019)	Native (Safe)	Leaf-wise	Medium/Large	Less tunable regularization than XGBoost; defaults to greedy growth.	Acceptable
XGBoost (Chen and Guestrin, 2016)	Native (Safe)	Level-wise	Any	None. Level-wise growth and L1/L2 regularization ideal for stability.	Best
CatBoost (Yandex, 2017)	Native (Safe)	Symmetric	Medium/Large	Slower training for pure numerical tasks; heavier dependency.	Alternative

Growth Strategies

- **Level-wise Growth (XGBoost):** Grows the tree layer-by-layer. Creates balanced trees, acting as a natural regularizer against experimental noise.
- **Leaf-wise Growth (LightGBM/HGB):** Splits the single leaf with the highest error. Can grow deep, lopsided trees that "memorize" outliers in small datasets.

1. Handles Missing Physics Natively (Sparsity Awareness)

- **The Problem:** Nuclear datasets are sparse and incomplete. Spin (J) or Parity (pi) are often unknown (missing).

- **The XGBoost Solution:** Unlike older algorithms that crash or require dangerous guessing (imputation), XGBoost treats missing values as information. It automatically learns the optimal path for "Unknown" data (e.g., if Spin is unknown, treat it as a potential match until proven otherwise).

2. Prevents Overfitting on Small Datasets (Regularization)

- **The Problem:** Nuclear level schemes are "small data" (typically less than 200 levels). Algorithms like LightGBM are designed for millions of rows and will aggressively "memorize" experimental noise in small datasets.
- **The XGBoost Solution:** Uses Level-wise growth (building balanced trees) rather than greedy Leaf-wise growth. Combined with built-in L1/L2 Regularization (mathematical penalties for complex models), it remains conservative and stable, prioritizing general physics trends over noise.

3. Enforces Physics Constraints (Statistical and Logical)

- **The Problem:** Standard models can violate physical laws. They might "learn" from noise that a high Z-score (e.g., 3-sigma difference) is better than a low Z-score, or they might "average out" a fatal selection rule violation (e.g., averaging a 90% energy match with a 0% spin match).
- **The XGBoost Solution:**
  - **Statistical Logic (Monotonicity):** Explicitly enforce Monotonic Constraints on the Z-score feature (|Delta E|/sigma). This mathematically guarantees that as statistical deviation increases, match probability must decrease, forcing the model to respect experimental uncertainties.
  - **Hard Vetoes (Boosting Strategy):** As a sequential learner, XGBoost handles binary exclusions (like Spin Parity vetoes) effectively. If an early tree predicts a match based on energy, a subsequent tree can detect the veto condition and apply a strong negative correction, driving the final probability to zero.

Level 5: The Implementation

Feature Engineering

Feature engineering is the process of transforming raw data into meaningful input features that enable the machine learning model to detect patterns, relationships, and interactions. Feature engineering is one of the most critical steps in building high-performance tree-based models.

Well-engineered features can significantly boost model performance, leading to improved accuracy and predictive power. XGBoost is capable of handling complex data relationships.

Feature_Engineer.py

Model Configuration

Level_Matcher.py