Gradient Boosted Decision Trees (GBDT)

Tutorial on Implementation and Interpretability in MoDeVa

July 5, 2025

Tutorial Overview:

- GBDT fundamentals and mathematical foundations
- Implementation with XGBoost, LightGBM, and CatBoost
- Interpretability through Functional ANOVA
- Special cases: GAM and GAMI structures
- Monotonicity constraints and domain knowledge
- Global and local interpretation techniques

Outline

What is Gradient Boosted Decision Trees?

Definition: GBDT is a powerful ensemble learning algorithm that builds a sequence of decision trees, where each subsequent tree is trained to correct the errors of its predecessors.

Key Characteristics:

- **Sequential Learning**: Trees built one after another
- **Error Correction**: Each tree learns from previous mistakes
- **Gradient-Based**: Uses gradients of loss function for optimization
- **Versatile**: Handles both regression and classification tasks

vs. Random Forests:

- GBDT: Sequential, dependent trees
- Random Forests: Parallel, independent trees

Why GBDT Matters

Advantages:

- **High Accuracy**: Often achieves state-of-the-art performance
- **Flexibility**: Works well with various data types
- **Feature Handling**: Robust to missing values and outliers
- **Interpretability**: Can be made interpretable with constraints

Applications:

- Credit scoring and risk assessment
- Sales forecasting and demand prediction
- Medical diagnosis and treatment planning
- Marketing response modeling

Popular Implementations:

- XGBoost: Performance and speed optimization
- LightGBM: Memory efficiency and fast training
- CatBoost: Categorical feature handling

GBDT Mathematical Framework

Overall Ensemble Model:

$$F_M(x) = F_0(x) + \sum_{m=1}^{M} \gamma_m T_m(x) \qquad (1)$$

Where:

- $F_0(x)$: Initial model (often constant, e.g., mean for regression)
- $T_m(x)$: Decision tree added at iteration m
- γ_m: Learning rate controlling tree contribution
- M: Total number of trees

Key Insight: Each tree adds a small correction to the ensemble, gradually improving predictions through iterative refinement.

Pseudo-Residual Calculation

Gradient-Based Learning:
At each iteration m, compute pseudo-residuals:

$$r_{im} = -\frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)} \tag{2}$$

Components:

- y_i: True target value for sample i
- $F_{m-1}(x_i)$: Current model prediction for sample i
- r_{im}: Pseudo-residual (negative gradient) for sample i
- L: Loss function (e.g., squared loss, log-loss)

Intuition: Pseudo-residuals point in the direction of steepest descent, indicating how to adjust predictions to reduce loss.

Model Update Rule

Sequential Update Process:
After fitting tree $h_m(x)$ to pseudo-residuals:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x) \tag{3}$$

Complete GBDT Algorithm:

1. Initialize: $F_0(x) = \arg\min_\gamma \sum_{i=1}^n L(y_i, \gamma)$
2. For $m = 1$ to M:
    - Compute pseudo-residuals: $r_{im} = -\frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)}$
    - Fit tree $h_m(x)$ to residuals
    - Find optimal step size: $\gamma_m = \arg\min_\gamma \sum_{i=1}^n L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))$
    - Update model: $F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$
3. Return final model: $F_M(x)$

MoDeVa GBDT Ecosystem

Unified Wrapper Architecture:
MoDeVa provides comprehensive wrappers around leading GBDT implementations:

XGBoost

- Speed optimization
- Advanced regularization
- Cross-validation built-in
- Monotonicity constraints

LightGBM

- Memory efficient
- Fast training speed
- Categorical features
- Large-scale datasets

CatBoost

- Categorical handling
- Overfitting resistant
- No preprocessing
- Symmetric trees

Benefits of Unified Interface:

- Consistent API across implementations
- Easy model comparison and switching
- Integrated interpretability tools
- Unified hyperparameter management

Data Setup and Preprocessing

Preparing Data for GBDT:

```
from modeva import DataSet

# Create dataset object
ds = DataSet()
# Load built-in dataset
ds.load(name="BikeSharing")

# Preprocessing pipeline
ds.scale_numerical(features=("cnt",), method="log1p")
ds.set_feature_type(feature="hr",
                    feature_type="categorical")
ds.set_feature_type(feature="mnth",
                    feature_type="categorical")
ds.scale_numerical(features=ds.feature_names_numerical,
                   method="standardize")
ds.set_inactive_features(features=("yr", "season", "temp"))
ds.preprocess()

# Split data
ds.set_random_split(test_ratio=0.2, random_state=42)
```

Model Configuration

Setting Up GBDT Models:
Regression Models:

```
from modeva.models import (MoLGBMRegressor,
                           MoXGBRegressor,
                           MoCatBoostRegressor)

# LightGBM
model_lgbm = MoLGBMRegressor(name="LGBM_model",
                              max_depth=2,
                              n_estimators=100)

# XGBoost
model_xgb = MoXGBRegressor(name="XGB_model",
                           max_depth=2,
                           n_estimators=100)

# CatBoost
model_cat = MoCatBoostRegressor(name="CBoost_model",
                                max_depth=2,
                                n_estimators=100)
```

Model Configuration

Classification Models:

```
from modeva.models import (MoLGBMClassifier,
                           MoXGBClassifier,
                           MoCatBoostClassifier)

# Similar syntax for classification tasks
model_clf = MoXGBClassifier(name="XGB_classifier",
                            max_depth=2,
                            n_estimators=100)
```

Model Training and Evaluation

Training Process:

```
# Train the model
model_gbdt.fit(ds.train_x, ds.train_y)

# Make predictions
train_pred = model_gbdt.predict(ds.train_x)
test_pred = model_gbdt.predict(ds.test_x)
```

Performance Assessment:

```
from modeva import TestSuite
# Create comprehensive test suite
ts = TestSuite(ds, model_gbdt)
# Performance metrics
result = ts.diagnose_accuracy_table()
print(result.table)
```

Model Training and Evaluation

Performance Metrics Include:

- Regression: RMSE, MAE, R^2, MAPE
- Classification: Accuracy, Precision, Recall, F1, AUC
- Cross-validation scores
- Training vs. validation performance

Making GBDT Interpretable

The Challenge: GBDT models are typically considered "black boxes"

The Solution: Functional ANOVA Decomposition
Mathematical Framework:

$$f(x) = \mu + \sum_j f_j(x_j) + \sum_{j<k} f_{jk}(x_j, x_k) + \dots \tag{4}$$

Components:

- μ: Global intercept (average prediction)
- $f_j(x_j)$: Main effects (individual feature contributions)
- $f_{jk}(x_j, x_k)$: Pairwise interaction effects
- Higher-order terms: Complex interactions (limited by tree depth)

Key Insight: Deep trees create complex interactions; shallow trees enable clearer interpretation.

Special Cases: GAM and GAMI Structures

Depth-1 Trees: GAM Structure

$$f(x) = \mu + \sum_j f_j(x_j) \tag{5}$$

- Each tree makes single split: Results in Generalized Additive Model
- Only main effects, no interactions
- Highly interpretable but limited flexibility

Depth-2 Trees: GAMI Structure

$$f(x) = \mu + \sum_j f_j(x_j) + \sum_{j<k} f_{jk}(x_j, x_k) \tag{6}$$

- Each tree makes up to two splits: GAM with Interactions (GAMI)
- Balances interpretability with interaction modeling
- Most practical choice for interpretable GBDT

ANOVA Decomposition Process

Two-Stage Process:

1. **Aggregation Stage**:
   - Start with tree ensemble: $f(x) = \sum_k \eta_k T_k(x)$
   - Decompose into leaf nodes:
     $f(x) = \sum_m v_m \prod_{j \in S_m} I(s^l_{mj} \leq x_j < s^u_{mj})$
   - Assign effects based on split variables:
     - Main effects: 1 split variable
     - Pairwise interactions: 2 split variables
     - Higher-order: More split variables

2. **Purification Stage**:
   - Ensure orthogonality: $\int f_{i_1 \ldots i_t} \, dx_k = 0$
   - Remove identifiability issues
   - Center effects to have zero mean
   - Cascade from high-order to low-order interactions

Feature Importance Analysis

Understanding Overall Feature Impact:

```
# Global feature importance
result = ts.interpret_fi()
result.plot()
```

Importance Metrics:

- Based on variance of marginal effects
- Normalized to sum to 1
- Accounts for feature scale differences
- Shows relative importance across features

Interpretation:

- Higher values indicate stronger influence on predictions
- Helps identify key drivers in the model
- Supports feature selection decisions
- Guides business focus areas

Effect Importance Analysis

ANOVA Component Analysis:

```
# Global effect importance
result = ts.interpret_ei()
result.plot()
```

Effect Types:

- **Main effects**: Individual feature contributions
- **Interaction effects**: Pairwise feature interactions
- **Higher-order**: Complex multi-feature interactions

Applications:

- Understand model complexity
- Identify significant interactions
- Validate domain knowledge
- Guide feature engineering

Effect Visualization

Main Effect Plots:

```
# Visualize main effect for specific feature
result = ts.interpret_effects(features="hr")
result.plot()
```

Plot Interpretation:

- Shows how feature affects predictions
- Reveals non-linear relationships
- Indicates feature importance regions
- Supports business understanding

Interaction Effect Plots:

```
# Visualize pairwise interaction
result = ts.interpret_effects(features=("temp", "hum"))
result.plot()
```

Individual Prediction Analysis

Local Feature Importance:

```
# Local interpretation for specific sample
result = ts.interpret_local_fi(sample_index=10,
                               centered=True)
result.plot()
```

Local Effect Importance:

```
# ANOVA-based local explanation
result = ts.interpret_local_ei(sample_index=10,
                               centered=True)
result.plot()
```

Local Explanation Components:

- Feature/effect contributions to specific prediction sample
- Comparison to average behavior
- Direction and magnitude of effects

Introduction to Monotonicity Constraints

What are Monotonicity Constraints?
Enforce that certain feature-response relationships follow domain knowledge:

Common Examples:

- **Credit Scoring**: Score should increase with income
- **Risk Assessment**: Risk should decrease with credit rating
- **Real Estate**: Property value should increase with square footage
- **Marketing**: Response rate should increase with previous purchases

Mathematical Constraint: For monotonic increasing feature x_j:

$$\frac{\partial f(x)}{\partial x_j} \geq 0 \quad \forall x \tag{7}$$

Implementation: Available in XGBoost through MoDeVa's `MoXGBRegressor` and `MoXGBClassifier`

Benefits of Monotonicity Constraints

Interpretability Enhancement:

- **Smoother Functions**: Reduces noise and fluctuations
- **Clearer Patterns**: More interpretable global trends
- **Consistent SHAP**: More stable local explanations
- **Business Logic**: Aligned with domain knowledge

Model Quality Improvements:

- **Reduced Overfitting**: Constraints act as regularization
- **Better Generalization**: More robust to new data
- **Stable Performance**: Often maintains or improves accuracy

Interaction with ANOVA:

- Guarantees monotonic shape for constrained features
- Preserves interpretability in decomposition
- Maintains partial monotonicity in interactions
- Simplifies effect visualization

Model Selection Guidelines

Choosing the Right GBDT Implementation:

Use XGBoost when:

- Need monotonicity constraints
- Want extensive hyperparameter control
- Performance is critical
- Working with structured data

Use LightGBM when:

- Large datasets (100K+ samples)
- Memory is limited
- Training speed matters
- Categorical features present

Use CatBoost when:

- Many categorical features
- Minimal preprocessing desired
- Overfitting is a concern
- New to GBDT

Tree Depth Selection:

- **Depth 1**: Maximum interpretability (GAM)
- **Depth 2**: Good balance (GAMI) - *recommended*
- **Depth 3+**: Higher performance, reduced interpretability

Hyperparameter Tuning Strategy

Key Parameters to Optimize:

1. **Tree Structure**:
   - `max_depth`: Start with 2, increase if needed
   - `n_estimators`: Start with 100, use early stopping
   - `min_child_weight`: Increase to prevent overfitting
2. **Learning Control**:
   - `learning_rate`: Start with 0.1, decrease for more trees
   - `subsample`: Try 0.8-1.0 for regularization
   - `colsample_bytree`: Use 0.8-1.0 for feature sampling
3. **Tuning Process**:
   1. Fix depth at 2 for interpretability
   2. Tune number of estimators with early stopping
   3. Optimize learning rate and regularization
   4. Apply monotonicity constraints if needed

Key Takeaways

MoDeVa Integration:

- Unified interface across XGBoost, LightGBM, and CatBoost
- Integrated interpretability through Functional ANOVA
- Easy model comparison and deployment

Interpretability:

- Depth constraints enable GAM/GAMI structures
- Global and local explanation methods
- Monotonicity constraints for domain knowledge integration

Best Practices:

- Start with depth-2 trees for interpretability
- Use early stopping and cross-validation
- Apply monotonicity constraints when appropriate
- Validate interpretations with domain experts