Linear Trees and Gradient Boosted Linear Trees

Combining Tree-Based Models with Linear Predictions

July 5, 2025

Introduction to Linear Trees and GBLT

- Linear Trees replace constant predictions at leaves with linear models
- Gradient Boosted Linear Trees (GBLT) extend gradient boosting by using Linear Trees as base learners
- Provides both local flexibility and interpretability
- Combines strengths of tree-based methods and linear models
- Powerful approach for datasets with local linear patterns

Linear Trees - Core Concept

Traditional Decision Trees

- Use constant predictions at leaf nodes
- Each leaf predicts the mean value of the target for data points in that leaf
- Limited in capturing linear relationships

Linear Trees

- Replace constants with linear regression models at each leaf
- Predict target as linear combination of features
- Each leaf has its own regression model

Key Benefits

- Handle local linear trends in the data
- Introduce flexibility to capture non-linear relationships through tree splits
- Better predictive performance for problems with local linearity

Gradient Boosted Linear Trees (GBLT) - Concept

Key Concepts

- Extends gradient boosting framework
- Uses Linear Trees as base learners
- Sequential ensemble building
- Each tree learns from the residuals of previous trees

Training Process

1. Initialize model with a constant value (global mean)
2. For each iteration:
   - Compute residuals/gradients from current model
   - Fit a Linear Tree to these residuals
   - Add the new tree to the ensemble with a weight
   - Update the model predictions
3. Continue until a stopping criterion is met

Mathematical Foundation - Ensemble Structure

Overall Model Equation

$$f(\mathbf{x}) = f_0 + \sum_{m=1}^{M} \gamma_m T_m(\mathbf{x}) \tag{1}$$

Components

- $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ - Feature vector
- f_0 - Baseline prediction (global mean)
- γ_m - Weight of the mth tree
- $T_m(\mathbf{x})$ - Prediction function of tree m

Ensemble Properties

- Additive model structure
- Each tree contributes incrementally
- Combines multiple weak learners
- Weights control contribution of each tree

Tree Structure and Prediction

Tree Prediction Function

$$T_m(\mathbf{x}) = \begin{cases} \beta_{m0}^{(L)} + \sum_{i=1}^{d} \beta_{mi}^{(L)} x_i, & \text{if } x_{j_m} \leq t_m \\ \beta_{m0}^{(R)} + \sum_{i=1}^{d} \beta_{mi}^{(R)} x_i, & \text{if } x_{j_m} > t_m \end{cases} \quad (2)$$

Tree Components

- x_{j_m} - Feature selected for splitting
- t_m - Threshold value for the split
- $\beta_{m0}^{(L)}, \beta_{m0}^{(R)}$ - Intercepts
- $\beta_{mi}^{(L)}, \beta_{mi}^{(R)}$ - Coefficients

Distinction from Standard Trees

- Standard trees have constant predictions
- Linear trees have linear models at leaves
- Can capture local linear trends
- More expressive at each leaf

Using Linear Trees in MoDeVa

```python
from modeva import DataSet
ds = DataSet()
ds.load(name="BikeSharing")
# Preprocessing steps
ds.scale_numerical(features=("cnt",), method="log1p")
ds.set_feature_type(feature="hr", feature_type="categorical"
    )
ds.preprocess()
ds.set_random_split()

# For regression tasks
from modeva.models import MoGLMTreeRegressor
model_glmt = MoGLMTreeRegressor(name="GLMT", max_depth=10)

# For classification tasks
from modeva.models import MoGLMTreeClassifier
model_glmt = MoGLMTreeClassifier(name="GLMT", max_depth=10)

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

Using Gradient Boosted Linear Trees in MoDeVa

```python
# For regression tasks
from modeva.models import MoGLMTreeBoostRegressor
model_gblt = MoGLMTreeBoostRegressor(name="GBLT",
                                     max_depth=1,
                                     n_estimators=100)

# For classification tasks
from modeva.models import MoGLMTreeBoostClassifier
model_gblt = MoGLMTreeBoostClassifier(name="GBLT",
                                      max_depth=1,
                                      n_estimators=100)

# Train model
model_gblt.fit(ds.train_x, ds.train_y)

# Create TestSuite for evaluation
from modeva import TestSuite
ts = TestSuite(ds, model_gblt)
```

Interpretability through Functional ANOVA

Decomposition of Prediction Function

$$f(\mathbf{x}) = f_0 + \sum_i f_i(x_i) + \sum_{i<j} f_{ij}(x_i, x_j) + \dots \tag{3}$$

Baseline: Global mean prediction

$$f_0 = \mathbb{E}[f(\mathbf{x})] \tag{4}$$

Main Effects: Individual feature contributions

$$f_i(x_i) = \mathbb{E}_{\mathbf{x}_{\setminus i}}[f(\mathbf{x}) \mid x_i] - f_0 \tag{5}$$

Interaction Effects: Joint feature effects

$$f_{ij}(x_i, x_j) = \mathbb{E}_{\mathbf{x}_{\setminus \{i,j\}}}[f(\mathbf{x}) \mid x_i, x_j] - f_i(x_i) - f_j(x_j) - f_0 \tag{6}$$

GBLT Implementation Process

1. **Train the Boosted Ensemble**
   Build model using depth-1 trees, record splitting feature x_{j_m}, threshold t_m, and linear models
2. **Aggregate Tree Predictions**
   $f(\mathbf{x}) = f_0 + \sum_{m=1}^{M} \gamma_m \, T_m(\mathbf{x})$
3. **Compute the Baseline**
   $f_0 = \mathbb{E}[f(\mathbf{x})]$
4. **Derive Main Effects**
   $f_i(x_i) = \mathbb{E}_{\mathbf{x}_{\setminus i}}\left[f(x_i, \mathbf{x}_{\setminus i})\right] - f_0$
5. **Extract Interaction Effects**
   $f_{ij}(x_i, x_j) = \mathbb{E}_{\mathbf{x}_{\setminus \{i,j\}}}\left[f(x_i, x_j, \mathbf{x}_{\setminus \{i,j\}})\right] - f_i(x_i) - f_j(x_j) - f_0$
6. **Interpret and Visualize**
   Use decomposition to gain insights into feature contributions and interactions

Global Model Understanding

Feature Importance

Impact of features on predictions

```
result = ts.interpret_fi()
result.plot()
```

Importance Metrics

- Based on variance of marginal effects
- Normalized to sum to 1
- Higher values indicate stronger influence
- Accounts for feature scale differences

Effect Importance

Contribution of ANOVA components

```
result = ts.interpret_ei()
result.plot()
```

Global Effect Plots

Visualize feature relationships

```
# Main effect
result = ts.
    interpret_effects(
    features="hr")
result.plot()
```

Understanding Individual Predictions

Local Feature Importance

Feature impact for specific samples

```
result = ts.
    interpret_local_fi(
     sample_index=10,
     centered=True)
result.plot()
```

Local Effect Importance

Effect contribution for individuals

```
result = ts.
    interpret_local_ei(
     sample_index=10,
     centered=True)
result.plot()
```

Centering Options

Uncentered Analysis (centered=False):

- Raw feature contributions
- Direct interpretation
- May have identifiability issues

Centered Analysis (centered=True):

- Compares to population mean
- More stable interpretation
- Better for relative importance

Advantages of Linear Trees

Local Flexibility

- Captures more nuanced patterns in the data
- Each leaf adapts to local relationships
- Finds region-specific linear trends

Improved Predictive Power

- Reduces bias in regions with linear relationships
- More expressive than standard trees
- Fewer splits needed to model complex functions

Interpretability

- Each leaf's linear model provides insight into feature contributions
- Coefficients have clear meaning within each region
- Maintains the structural interpretability of trees

Advantages of Gradient Boosted Linear Trees

Enhanced Predictive Performance

Combines boosting with local linear models

Balanced Complexity

More expressive than standard trees, more structured than black-box models

Rich Interpretation Framework

Functional ANOVA provides detailed insights

Handles Complex Relationships

Captures both global and local patterns

Comprehensive Tooling

Extensive interpretation methods in MoDeVa

Enhanced Robustness

Less prone to overfitting in regions with linear relationships

When to Use Gradient Boosted Linear Trees

Ideal Use Cases

- Datasets with both global trends and local linear relationships
- Applications requiring both accuracy and interpretability
- When insights about feature interactions are important
- Problems with mixed categorical and numerical features

Industries and Applications

- **Finance**: Risk modeling, credit scoring
- **Healthcare**: Patient outcome prediction
- **Marketing**: Customer response modeling
- **Energy**: Demand forecasting