Tree Machine

PYTHON 3.12 PYTHON 3.13 LICENSE MIT

Tree Machine is a production-ready Python library that provides an AutoML companion for fitting tree models with ease. Built on top of gradient-boosting backends (XGBoost, CatBoost) with Bayesian optimization, it offers a unified interface for classification, regression, and quantile regression tasks while maintaining scikit-learn compatibility.

Unlike traditional ML libraries that require extensive hyperparameter tuning, Tree Machine specializes in automated model selection and optimization—using Bayesian optimization to find the best model and hyperparameters while providing enterprise-grade features like SHAP explanations, monotonicity constraints, and interaction controls.

✨ Key Features

- 🤖 **Automated Hyperparameter Optimization**: Bayesian optimization with Optuna for intelligent parameter search
- 🎯 **Multi-Task Support**: Classification, regression, and quantile regression in one unified API
- 🔒 **Production Constraints**: Built-in monotonicity constraints and feature interaction controls
- 📊 **SHAP Integration**: Automatic model explanations with TreeExplainer
- 🎨 **Scikit-Learn Compatible**: Drop-in replacement that works with existing pipelines
- ⚡ **Multi-Backend Boosting**: Switch between XGBoost and CatBoost

🚀 Quick Start

```
from tree_machine import ClassifierCV, RegressionCV, QuantileCV,
default_classifier
from sklearn.model_selection import StratifiedKFold
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

# Generate sample data
X, y = make_classification(n_samples=1000, n_features=20, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Create and fit an auto-tuned classifier
classifier = ClassifierCV(
    metric="roc_auc",
    cv=StratifiedKFold(n_splits=5),
    n_trials=50,
    timeout=300,  # 5 minutes
    config=default_classifier,
    backend="xgboost",  # "xgboost" (default) or "catboost"
)

# Fit with automatic hyperparameter optimization
classifier.fit(X_train, y_train)

# Make predictions
predictions = classifier.predict(X_test)
probabilities = classifier.predict_proba(X_test)

# Get SHAP explanations
explanations = classifier.explain(X_test)

# Access optimization results
print(f"Best parameters: {classifier.best_params_}")
print(f"CV scores: {classifier.cv_results}")
```

Regression Example

```
from tree_machine import RegressionCV, default_regression
from sklearn.model_selection import KFold
from sklearn.datasets import make_regression

# Generate regression data
X, y = make_regression(n_samples=1000, n_features=20, noise=0.1, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Create auto-tuned regressor
regressor = RegressionCV(
    metric="neg_mean_squared_error",
    cv=KFold(n_splits=5),
    n_trials=50,
    timeout=300,
    config=default_regression
)

regressor.fit(X_train, y_train)
predictions = regressor.predict(X_test)
```

Quantile Regression Example

```
from tree_machine import QuantileCV

# Predict the 90th percentile
quantile_regressor = QuantileCV(
    alpha=0.9,
    cv=KFold(n_splits=5),
    n_trials=50,
    timeout=300,
    config=default_regression
)

quantile_regressor.fit(X_train, y_train)
quantile_predictions = quantile_regressor.predict(X_test)
```

Installation

Install from GitHub

```
pip install git+https://github.com/vitorbezzan/tree_machine.git@main
```

Advanced Features

Monotonicity Constraints

```
from tree_machine import ClassifierCVConfig, OptimizerParams

# Create custom config with monotonicity constraints
custom_config = ClassifierCVConfig(
    monotone_constraints={"feature_1": 1, "feature_2": -1},  # increasing,
decreasing
    interactions=[["feature_1", "feature_3"]],  # allow only these interactions
    n_jobs=-1,
    parameters=OptimizerParams(),
    return_train_score=True,
)

classifier = ClassifierCV(
    metric="roc_auc",
    cv=StratifiedKFold(n_splits=5),
    n_trials=100,
    timeout=600,
    config=custom_config
)
```

Custom Optimization Parameters

```
from tree_machine import OptimizerParams

# Define custom parameter search space
custom_params = OptimizerParams(
    n_estimators=(100, 1000),
    max_depth=(3, 10),
    learning_rate=(0.01, 0.3),
    subsample=(0.8, 1.0),
    colsample_bytree=(0.8, 1.0),
    min_child_weight=(1, 10),
    gamma=(0.0, 1.0),
    reg_alpha=(0.0, 1.0),
    reg_lambda=(0.0, 1.0)
)
```

Pre-configured Setups

```python
from tree_machine import balanced_classifier, balanced_regression

# Use balanced configuration for better generalization
balanced_clf = ClassifierCV(
    metric="f1",
    cv=StratifiedKFold(n_splits=5),
    n_trials=50,
    timeout=300,
    config=balanced_classifier,  # Pre-configured for balanced performance
    backend="catboost",          # Swap to default "xgboost" as needed
)
```

Development Setup

To set up Tree Machine for development, follow these detailed steps:

1. Clone the Repository

```bash
git clone https://github.com/vitorbezzan/tree_machine.git
cd tree_machine
```

2. Create a Virtual Environment (Recommended)

```bash
# Using venv
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate

# Or using conda
conda create -n tree_machine python=3.12+
conda activate tree_machine
```

3. Install in Development Mode

This project uses `pyproject.toml` for dependency management. Install the package in editable mode with all dependencies:

```bash
# Install using Make (recommended)
make install

# Or install manually in development mode
pip install -e ".[dev]"
```

4. Verify Installation

```
# Run tests to verify everything is working
pytest

# Or run tests with coverage
pytest --cov=src --cov-report html
```

5. Development Workflow

- The source code is located in `src/tree_machine/`
- Tests are in the `tests/` directory
- Documentation files are in `docs/`
- Use `pytest` to run tests during development
- The project configuration is managed through `pyproject.toml`
- Pre-commit hooks are available for code quality

Requirements

- **Python**: 3.12+
- **Core Dependencies**:
- numpy >= 2.0.0
- pandas >= 2.0.0
- scikit-learn >= 1.5.0
- xgboost >= 2.0.0
- catboost >= 1.2.8
- optuna >= 4.1.0
- pydantic >= 2.10.0
- shap >= 0.46.0

License

See LICENSE for details.