TurboCat

Technical Documentation v0.2.0

Next-Generation Gradient Boosting Library

Version	0.2.0
License	MIT
Language	C++ / Python
Platform	Linux, macOS, Windows

Table of Contents

1. Introduction

TurboCat is a high-performance gradient boosting library written in C++ with Python bindings. It implements state-of-the-art research techniques to achieve quality comparable to CatBoost while being 3-10x faster in both training and inference.

Key Innovations

- **GradTree (AAAI 2024)** — Gradient-based global tree optimization
- **Robust Focal Loss** — Better handling of class imbalance
- **Tsallis Entropy Splitting** — Non-extensive entropy for splits
- **LDAM Loss** — Label-distribution-aware margin loss
- **GOSS Sampling** — Gradient-based One-Side Sampling
- **SIMD Optimizations** — AVX2/AVX-512 vectorized histograms

2. Architecture Overview

Project Structure

```
turbocat/
■■■ include/turbocat/    # C++ headers
■   ■■■ types.hpp        # Core types (Float, Index)
■   ■■■ config.hpp       # Configuration structures
■   ■■■ dataset.hpp      # Dataset handling
■   ■■■ histogram.hpp    # Histogram builder
■   ■■■ tree.hpp         # Tree structures
■   ■■■ loss.hpp         # Loss functions
■   ■■■ booster.hpp      # Main booster class
■■■ src/
■   ■■■ core/            # Core implementations
■   ■■■ tree/            # Tree building (histogram.cpp, tree.cpp)
■   ■■■ boosting/        # Boosting logic (booster.cpp, loss.cpp)
■   ■■■ utils/           # SIMD, threading utilities
■■■ python/              # Python bindings (pybind11)
■■■ build/               # Build output (_turbocat.so)
```

Data Flow

Training pipeline: Raw Data → Binning (quantization to 0-255) → Histogram Building → Split Finding → Tree Construction → Gradient Update → Repeat for n_estimators.

Memory Layout

Column-major storage for features during training (cache-friendly for histogram building). Histograms use Structure-of-Arrays (SoA) for SIMD efficiency: separate arrays for grad, hess, count.

3. Installation

Requirements

Requirement	Version	Notes
C++ Compiler	C++17+	GCC 10+, Clang 12+, Apple Clang 14+
CMake	3.18+	Build system
Python	3.8+	For Python bindings
NumPy	1.19+	Required for Python API
OpenMP	Optional	For parallel training (recommended)
Eigen3	Auto	Downloaded automatically by CMake

Build from Source

```
git clone https://github.com/ispromadhka/Turbo-Cat.git
cd Turbo-Cat
mkdir build && cd build
cmake .. -DCMAKE_BUILD_TYPE=Release
make -j8
# Python module: build/_turbocat.cpython-3XX-*.so
```

CMake Options

Option	Default	Description
CMAKE_BUILD_TYPE	Release	Build type (Debug/Release)
TURBOCAT_BUILD_TESTS	ON	Build unit tests
TURBOCAT_USE_OPENMP	ON	Enable OpenMP parallelization
TURBOCAT_USE_AVX2	Auto	Enable AVX2 SIMD

4. Quick Start

```python
import sys
sys.path.insert(0, 'build')  # Path to compiled module
import _turbocat as tc
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, roc_auc_score

# Generate data
X, y = make_classification(n_samples=10000, n_features=20, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Create classifier
model = tc.TurboCatClassifier(
    n_estimators=50,      # Number of trees
    max_depth=8,          # Maximum tree depth
    learning_rate=0.1,    # Step size
    verbosity=0           # Silent mode
)

# Train (IMPORTANT: convert to float32)
model.fit(X_train.astype(np.float32), y_train.astype(np.float32))

# Predict (wrap in np.array)
proba = np.array(model.predict_proba(X_test.astype(np.float32)))
predictions = (proba > 0.5).astype(int)

print(f"Accuracy: {accuracy_score(y_test, predictions):.4f}")
print(f"ROC-AUC:  {roc_auc_score(y_test, proba):.4f}")
```

5. C++ Core Components

5.1 Types (types.hpp)

```
namespace turbocat {
    using Float = float;              // Primary floating point type
    using Index = int32_t;            // Sample/row index
    using FeatureIndex = uint16_t;    // Feature/column index (max 65535)
    using BinIndex = uint8_t;         // Histogram bin (0-255)
    using TreeIndex = uint16_t;       // Node index in tree

    // Gradient pair for histogram accumulation
    struct GradientPair {
        Float grad = 0.0f;    // Gradient sum
        Float hess = 0.0f;    // Hessian sum
        uint32_t count = 0;   // Sample count

        GradientPair& operator+=(const GradientPair& other);
        GradientPair operator-(const GradientPair& other) const;
    };
}
```

5.2 Dataset (dataset.hpp)

```
class Dataset {
public:
    void from_dense(const Float* data, Index n_samples,
                    FeatureIndex n_features, const Float* labels);
    void compute_bins(const Config& config);  // Quantize to 0-255

    const BinnedData& binned() const;         // Column-major binned features
    const Float* gradients() const;
    const Float* hessians() const;
    void set_gradients(AlignedVector<Float>&& grads, AlignedVector<Float>&& hess);

    Index n_samples() const;
    FeatureIndex n_features() const;
};
```

5.3 Histogram Builder (histogram.hpp)

SIMD-optimized histogram construction - the performance-critical component:

```
class Histogram {
public:
    Histogram(FeatureIndex n_features, BinIndex max_bins = 255);
    void clear();
    GradientPair* bins(FeatureIndex feature);
    void subtract_from(const Histogram& parent, const Histogram& other);
};

class CPUHistogramBuilder : public HistogramBuilder {
public:
    void build(const Dataset& dataset,
               const std::vector<Index>& sample_indices,
               const std::vector<FeatureIndex>& feature_indices,
               Histogram& output) override;
private:
    void build_feature_avx2(...);   // 8x unrolling + prefetch
};
```

Key Optimizations:

• Feature-parallel building - each thread processes different features, no sync needed

• 8x loop unrolling - better instruction-level parallelism

• Prefetching - _mm_prefetch for next batch reduces cache misses

• Histogram subtraction - derive sibling in O(bins) instead of O(samples)

5.4 Tree (tree.hpp)

```
struct TreeNode {
    FeatureIndex split_feature = 0;  // Feature used for split
    BinIndex split_bin = 0;          // Bin threshold
    TreeIndex left_child = 0;
    TreeIndex right_child = 0;
    Float value = 0.0f;              // Leaf value (prediction)
    Float gain = 0.0f;               // Split gain
    uint8_t is_leaf = 1;
    GradientPair stats;              // Node statistics
};

class Tree {
public:
    void build(const Dataset& dataset,
               const std::vector<Index>& sample_indices,
               HistogramBuilder& hist_builder);
    Float predict(const Float* features, FeatureIndex n_features) const;
    std::vector<Float> feature_importance() const;
};
```

5.5 Loss Functions (loss.hpp)

Loss Type	Use Case	Key Property
LogLoss	Binary classification	Standard log loss
Focal	Imbalanced data	Down-weights easy examples
LDAM	Margin-aware	Class-dependent margins
Tsallis	Non-extensive	Generalized entropy

5.6 Booster (booster.hpp)

```
class Booster {
public:
    Booster();
    explicit Booster(const Config& config);

    void train(Dataset& train_data, Dataset* valid_data = nullptr);

    void predict_raw(const Dataset& data, Float* output, int n_trees = -1) const;
    void predict_proba(const Dataset& data, Float* output, int n_trees = -1) const;
    Float predict_single(const Float* features, FeatureIndex n_features) const;

    size_t n_trees() const;
    void save(const std::string& path) const;
    static Booster load(const std::string& path);
};
```

6. Python API

```
class TurboCatClassifier:
    """TurboCat Gradient Boosting Classifier

    Parameters
    ----------
    n_estimators : int, default=100 - Number of boosting iterations
    learning_rate : float, default=0.1 - Step size shrinkage
    max_depth : int, default=6 - Maximum depth of each tree
    max_bins : int, default=255 - Number of histogram bins
    min_child_weight : float, default=1.0 - Minimum hessian in leaf
    subsample : float, default=1.0 - Row sampling ratio
    colsample_bytree : float, default=1.0 - Feature sampling ratio
    lambda_l2 : float, default=1.0 - L2 regularization
    verbosity : int, default=1 - 0=silent, 1=progress
    """

    def fit(self, X, y):
        """Train the model. X, y must be float32 numpy arrays."""

    def predict_proba(self, X):
        """Predict probabilities. Returns list, wrap with np.array()."""

    def predict(self, X):
        """Predict class labels."""

    def feature_importance(self):
        """Return feature importance scores."""

    def save(self, path):
        """Save model to file."""

    @staticmethod
    def load(path):
        """Load model from file."""
```

Important Notes:

• Always convert arrays to float32: X.astype(np.float32)

• predict_proba returns list - wrap with np.array()

• Add build directory to path: sys.path.insert(0, 'build')

7. Configuration Parameters

7.1 Boosting Parameters

Parameter	Type	Default	Description
n_estimators	int	100	Number of trees
learning_rate	float	0.1	Step size (0.01-0.3)
subsample	float	1.0	Row sampling ratio
colsample_bytree	float	1.0	Feature sampling ratio
early_stopping	int	50	Patience for early stop

7.2 Tree Parameters

Parameter	Type	Default	Description
max_depth	int	6	Maximum tree depth (1-15)
max_bins	int	255	Histogram bins (max 255)
min_samples_leaf	int	20	Minimum samples in leaf
min_child_weight	float	1.0	Minimum hessian sum
lambda_l2	float	1.0	L2 regularization
lambda_l1	float	0.0	L1 regularization

7.3 Recommended Configurations

For Speed (production):

```
model = tc.TurboCatClassifier(n_estimators=50, max_depth=6,
                                learning_rate=0.1, verbosity=0)
```

For Quality (competition):

```
model = tc.TurboCatClassifier(n_estimators=200, max_depth=8,
                                learning_rate=0.05, subsample=0.8,
                                colsample_bytree=0.8)
```

For Imbalanced Data:

```
model = tc.TurboCatClassifier(n_estimators=100, max_depth=8,
                                learning_rate=0.1, min_child_weight=0.1)
```

8. Benchmark Results

Comprehensive benchmark on 30 datasets vs CatBoost:

8.1 Quality Comparison

Metric	TurboCat	CatBoost	p-value	Result
Accuracy	0.9164	0.9171	0.87	Tie
ROC-AUC	0.9515	0.9568	0.17	Tie
F1 Score	0.8786	0.8695	0.31	TurboCat
Recall	0.8657	0.8592	0.45	TurboCat

No statistically significant difference (p > 0.05). Quality parity achieved.

8.2 Performance

Metric	Result	Range
Training speedup	3.5x faster (mean)	Up to 18.9x
Inference speedup	9.7x faster (mean)	Up to 33x
Training wins	23/30 datasets	77%
Inference wins	30/30 datasets	100%

8.3 Results by Category

Category	TC Wins Acc	Train Speed	Infer Speed
Imbalanced	4/4	1.8x	5.7x
Synthetic	1/5	1.3x	7.3x
Scale (5K-50K)	2/3	5.3x	9.5x
High-Dim	2/4	7.1x	17.1x
Special	3/4	2.0x	15.1x

9. Strengths & Weaknesses

9.1 Strengths

Imbalanced Data - Key Advantage:

Imbalance	TC Recall	CB Recall	TC F1	CB F1
70/30	91.2%	87.4%	93.6%	91.3%
85/15	84.7%	75.9%	89.8%	84.7%
95/5	54.5%	45.5%	70.2%	62.1%
99/1	15.8%	3.5%	27.3%	6.8%

On extreme imbalance (99/1): 4x better F1 score!

Other Strengths:

• Speed - 3-10x faster training, up to 33x faster inference

• Medium-Large Scale - Best on 5K-50K samples

• Correlated Features - +0.2% ROC-AUC

• Data with Outliers - +0.3% ROC-AUC

9.2 Weaknesses

• Noisy Data - Up to -9.9% ROC-AUC with >10% label noise

• Small Datasets - CatBoost generalizes better on <1K samples

• High-Dim Sparse - Slightly worse with many irrelevant features

9.3 When to Use

Recommended:

■ Fraud detection, medical diagnosis (imbalanced classes)

■ Production deployment (fast inference required)

■ Real-time predictions

■ Medium-large datasets (5K+ samples)

Consider Alternatives:

■■ Very noisy data (>10% label noise)

■■ Very small samples (<500)

■■ Extreme high-dim sparse data

10. Advanced Usage

10.1 Cross-Validation

```
from sklearn.model_selection import StratifiedKFold

def cv_turbocat(X, y, n_splits=5, **params):
    skf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=42)
    scores = []
    for train_idx, val_idx in skf.split(X, y):
        X_tr, X_val = X[train_idx], X[val_idx]
        y_tr, y_val = y[train_idx], y[val_idx]

        model = tc.TurboCatClassifier(**params)
        model.fit(X_tr.astype(np.float32), y_tr.astype(np.float32))
        proba = np.array(model.predict_proba(X_val.astype(np.float32)))
        scores.append(roc_auc_score(y_val, proba))
    return np.mean(scores), np.std(scores)

mean, std = cv_turbocat(X, y, n_estimators=50, max_depth=8)
print(f"CV ROC-AUC: {mean:.4f} +/- {std:.4f}")
```

10.2 Feature Importance

```
# Get feature importance
importance = model.feature_importance()

# Plot
import matplotlib.pyplot as plt
plt.barh(range(len(importance)), importance)
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```

11. Troubleshooting

ImportError: No module named '_turbocat'

Add build directory to Python path:

```
import sys
sys.path.insert(0, '/path/to/turbocat/build')
import _turbocat as tc
```

TypeError: incompatible array type

Convert arrays to float32:

```
X = X.astype(np.float32)
y = y.astype(np.float32)
```

predict_proba returns list

Wrap result in np.array():

```
proba = np.array(model.predict_proba(X_test))
```

Slow training

Check OpenMP is enabled:

```
# Look for in CMake output:
# -- TurboCat: OpenMP support enabled

# If not, install:
# macOS: brew install libomp
# Linux: apt install libomp-dev
```

Support

GitHub: https://github.com/ispromadhka/Turbo-Cat