TailRisk

Risk-Aware Machine Learning
for Tail Risk Modeling

Technical Documentation

Version 0.1.2

Vishal Lakshmi Narayanan
lvishal1607@gmail.com
GitHub Repository

November 22, 2025

Abstract

TailRisk is a Python package designed for building machine learning models that excel at predicting extreme outcomes in insurance claims, financial losses, and other tail-risk scenarios. Traditional machine learning models optimize for average performance (MSE, MAE), which fails catastrophically when predicting rare, extreme events. This package implements novel methodologies including Loss-at-Risk (LaR) regression, CVaR-weighted ensembles, and hybrid meta-learning to address these critical challenges in risk modeling.

Keywords: Tail Risk, Machine Learning, Insurance, Finance, CVaR, Extreme Value Prediction, Quantile Regression

Contents

1 Introduction

1.1 The Tail Risk Problem

Traditional machine learning models optimize metrics such as Mean Squared Error (MSE) or Mean Absolute Error (MAE). While these metrics work well for average performance, they treat all prediction errors equally. This approach is **fundamentally flawed** for tail risk applications:

- **Insurance claims**: Predicting \$500 for a claim that costs \$50,000 creates severe financial exposure
- **Financial risk**: Underestimating tail risk leads to inadequate capital reserves (2008 financial crisis)
- **Healthcare costs**: Missing catastrophic cases can bankrupt risk pools

1.2 Why MSE Fails

Consider the mathematical formulation of MSE:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{1}$$

In this formulation, a \$10,000 error on a \$500 claim receives the *same weight* as a \$10,000 error on a \$100,000 claim. For tail risk applications, this is catastrophic.

1.3 The TailRisk Solution

TailRisk implements three key innovations:

1. **Loss-at-Risk (LaR) Regression**: Weighted regression with adaptive importance based on claim magnitude
2. **Hybrid Meta-Learning**: Two-stage architecture combining quantile regression and LaR optimization
3. **Tail-Focused Metrics**: CVaR, Tail Coverage Ratio, and Detection Rate for proper evaluation

2 Installation

2.1 Requirements

- Python 3.8 or higher
- NumPy $\geq$ 1.21.0
- Pandas $\geq$ 1.3.0
- Scikit-learn $\geq$ 1.0.0
- SciPy $\geq$ 1.7.0
- Matplotlib $\geq$ 3.4.0

2.2 Installation via PyPI

The recommended installation method is via pip:

```
pip install tailrisk
```

2.3 Installation from Source

For development or latest features:

```
git clone https://github.com/VishalLakshmiNarayanan/TailriskLib.git
cd TailriskLib
pip install -e .
```

2.4 Verification

Verify successful installation:

```python
import tailrisk
print(f"TailRisk version: {tailrisk.__version__}")
```

3 Quick Start

3.1 Basic Example: LaR Regressor

```
import numpy as np
from sklearn.model_selection import train_test_split
from tailrisk import LaRRegressor, tail_validation_summary

# Generate heavy-tailed data
X = np.random.randn(1000, 10)
y = np.random.exponential(scale=1000, size=1000)

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Train LaR model
model = LaRRegressor(alpha=2.0)
model.fit(X_train, y_train)

# Predict and evaluate
predictions = model.predict(X_test)
metrics = tail_validation_summary(y_test, predictions)

print(f"CVaR(95%): ${metrics['cvar_95']:,.2f}")
print(f"Tail Coverage Ratio: {metrics['tcr_99']:.3f}")
```

3.2 Advanced Example: Hybrid Meta-Learner

```
from tailrisk import HybridMetaLearner
from sklearn.ensemble import (
    RandomForestRegressor,
    GradientBoostingRegressor
)

# Define diverse base models
base_models = [
    ('rf', RandomForestRegressor(
        n_estimators=100,
        max_depth=10,
        random_state=42
    )),
    ('gb', GradientBoostingRegressor(
        n_estimators=100,
        max_depth=5,
        random_state=42
    ))
]

# Create hybrid meta-learner
model = HybridMetaLearner(
    base_estimators=base_models,
    quantile=0.95,           # Focus on 95th percentile
    blend_lambda=0.25,       # 25% quantile, 75% LaR
    lar_alpha=1.5,           # LaR weighting strength
    cv_folds=5               # Cross-validation folds
)
```

```
# Train and predict
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

4 API Reference

4.1 Models

4.1.1 LaRRegressor

Class: `tailrisk.LaRRegressor`
Loss-at-Risk weighted regression model.

Parameters:

- `alpha` : *float*, default=2.0
  - Weight scaling factor
  - Higher values increase focus on large claims
  - Recommended range: 1.0 to 3.0
- `base_estimator` : *estimator*, default=None
  - Base regression model
  - Must support `sample_weight` parameter
  - If None, uses `LinearRegression()`

Methods:

- `fit(X, y, sample_weight=None)` : Fit the model
- `predict(X)` : Generate predictions

Attributes:

- `base_estimator_` : Fitted base estimator

Mathematical Formulation:
The LaR objective function is defined as:

$$\mathcal{L}_{\text{LaR}} = \frac{1}{n}\sum_{i=1}^{n} w_i (y_i - \hat{y}_i)^2 \tag{2}$$

where the weights are:

$$w_i = 1 + \alpha \cdot \frac{y_i}{\max(y)} \tag{3}$$

This ensures larger target values receive proportionally higher importance during optimization.

4.1.2 HybridMetaLearner

Class: `tailrisk.HybridMetaLearner`

Advanced two-stage meta-learning ensemble for tail risk prediction.

Architecture:

1. **Stage 1**: Generate meta-features using cross-validated predictions from diverse base models
2. **Stage 2a**: Train quantile regression meta-model focusing on high quantiles
3. **Stage 2b**: Optimize LaR-weighted combination of base model predictions
4. **Stage 3**: Blend quantile and LaR predictions for final output

Parameters:

- `base_estimators` : *list of (str, estimator) tuples*
  - List of base models to ensemble
  - Recommended: Use diverse model types (trees, linear, boosting)
- `quantile` : *float*, default=0.95
  - Target quantile for quantile regression
  - Range: 0.90 to 0.99
- `blend_lambda` : *float*, default=0.25
  - Blending weight: $\hat{y} = \lambda \cdot \hat{y}_{\text{quantile}} + (1 - \lambda) \cdot \hat{y}_{\text{LaR}}$
  - Range: 0.0 to 1.0
  - Recommended: 0.2 to 0.3
- `lar_alpha` : *float*, default=1.5
  - LaR weight scaling factor
  - Range: 1.0 to 2.0
- `cv_folds` : *int*, default=5
  - Number of cross-validation folds
  - Typical range: 3 to 10

Final Prediction Formula:

$$\hat{y}_{\text{final}} = \lambda \cdot \hat{y}_{\text{quantile}} + (1 - \lambda) \cdot \hat{y}_{\text{LaR}} \tag{4}$$

4.1.3 CVaRWeightedEnsemble

Class: `tailrisk.CVaRWeightedEnsemble`
Ensemble that weights models based on inverse CVaR performance.

Parameters:

- `estimators` : *list of (str, estimator) tuples*
- `alpha` : *float*, default=0.95
  - CVaR percentile for weight calculation
  - Models with lower CVaR receive higher weight

4.2 Metrics

4.2.1 Conditional Value-at-Risk (CVaR)

Function: `tailrisk.cvar_loss(y_true, y_pred, alpha=0.95)`

Definition:
CVaR measures the average squared error in the worst $(1-\alpha)\%$ of predictions:

$$\text{CVaR}_\alpha = \mathbb{E}[(Y-\hat{Y})^2 \mid (Y-\hat{Y})^2 \geq \text{VaR}_\alpha] \tag{5}$$

where VaR_α is the Value-at-Risk at level α.

Parameters:

- `y_true` : array-like, True target values
- `y_pred` : array-like, Predicted values
- `alpha` : float, Percentile threshold (0.90–0.99)

Returns:

- `cvar` : float, CVaR metric (lower is better)

Interpretation:

- Lower values indicate better tail risk performance
- Industry standard in finance and insurance
- Focuses exclusively on worst-case scenarios

4.2.2 Tail Coverage Ratio (TCR)

Function: `tailrisk.tail_coverage_ratio(y_true, y_pred, quantile=0.99)`

Definition:
TCR measures the fraction of extreme tail value captured by predictions:

$$\text{TCR}_q = \frac{\sum_{i\in\mathcal{T}_q} \hat{y}_i}{\sum_{i\in\mathcal{T}_q} y_i} \tag{6}$$

where $\mathcal{T}_q$ is the set of indices where $y_i > q$-th percentile.

Interpretation:

- TCR $= 1.0$: Perfect coverage (100% of tail value captured)
- TCR < 1.0 : Underprediction (dangerous for risk management)
- TCR > 1.0 : Overprediction (conservative, may be acceptable)

4.2.3 Detection Rate

Function: `tailrisk.detection_rate(y_true, y_pred, quantile=0.95)`

Definition:
Percentage of actual extreme cases correctly predicted as extreme:

$$\text{Detection Rate}_q = \frac{|\{i : y_i > q_y \wedge \hat{y}_i > q_{\hat{y}}\}|}{|\{i : y_i > q_y\}|} \tag{7}$$

where q_y and $q_{\hat{y}}$ are the q-th percentiles of y and $\hat{y}$, respectively.

Use Case:
Critical for early warning systems and reserve planning.

4.2.4 tail_validation_summary

Function: `tailrisk.tail_validation_summary(y_true, y_pred)`
Returns comprehensive dictionary of all tail risk metrics:

- `mse_overall` : Overall mean squared error
- `mse_extreme` : MSE on extreme values (> 95th percentile)
- `cvar_90`, `cvar_95`, `cvar_99` : CVaR at different thresholds
- `tcr_95`, `tcr_99` : Tail coverage ratios
- `detection_90`, `detection_95`, `detection_99` : Detection rates
- `lar` : Loss-at-Risk metric

5 Methodology

5.1 The Hybrid Meta-Learning Framework

The HybridMetaLearner implements a novel architecture designed to optimize tail risk prediction:

5.1.1 Stage 1: Diverse Base Models

Train multiple heterogeneous models:

$$\mathcal{M} = \{f_1, f_2, \ldots, f_k\} \tag{8}$$

Recommended model types:

- Tree-based: Random Forests, Gradient Boosting
- Linear: Ridge, Lasso
- Ensemble: XGBoost, LightGBM

5.1.2 Stage 2a: Quantile Meta-Model

Use quantile regression on meta-features:

$$\hat{f}^{(\alpha)}_{\text{quantile}} = \arg\min_f \sum_{i=1}^{n} \rho_\alpha(y_i - f(\mathbf{z}_i)) \tag{9}$$

where ρ_α is the pinball loss function:

$$\rho_\alpha(u) = u(\alpha - \mathbb{I}(u < 0)) \tag{10}$$

and $\mathbf{z}_i$ are meta-features from cross-validated base model predictions.

5.1.3 Stage 2b: LaR-Weighted Optimization

Optimize weights for base model combination:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} w_i \left(y_i - \sum_{j=1}^{k} w_j f_j(\mathbf{x}_i) \right)^2 \tag{11}$$

subject to $\sum_{j=1}^{k} w_j = 1$ and $w_j \geq 0$.

5.1.4 Stage 3: Hybrid Blending

Final prediction combines both approaches:

$$\hat{y}_{\text{final}} = \lambda \cdot \hat{y}_{\text{quantile}} + (1 - \lambda) \cdot \hat{y}_{\text{LaR}} \tag{12}$$

Rationale:

- Quantile model excels at *detecting* extreme events
- LaR model maintains overall *accuracy*
- Blending provides optimal balance (typically $\lambda = 0.25$)

6 Use Cases

6.1 Insurance Claims Prediction

Problem Statement:

Catastrophic claims (top 1% of cases) often represent 30–50% of total losses. Traditional models severely underestimate these claims, leading to:

- Inadequate reserves
- Insolvency risk
- Incorrect premium pricing

TailRisk Solution:

```python
from tailrisk import HybridMetaLearner
from sklearn.ensemble import RandomForestRegressor

# Insurance-specific configuration
model = HybridMetaLearner(
    base_estimators=[
        ('rf', RandomForestRegressor(n_estimators=200)),
        # Add domain-specific models
    ],
    quantile=0.99,       # Focus on top 1% of claims
    blend_lambda=0.20    # Prioritize accuracy
)

model.fit(X_train, y_train)
claim_predictions = model.predict(X_test)
```

Results:

Typical improvements over baseline models:

- TCR@99% improves from 0.058 to 0.102 (+76%)
- CVaR(95%) reduces by 10–15%
- Detection Rate@95% increases from 0.8% to 2.5%

6.2 Financial Risk Management

Problem Statement:

Value-at-Risk (VaR) models systematically underestimate tail risk, as evidenced during the 2008 financial crisis. Regulatory frameworks (Basel III) now require CVaR-based risk assessment.

TailRisk Solution:

```python
from tailrisk import LaRRegressor
from tailrisk.metrics import cvar_loss

# Financial loss prediction
model = LaRRegressor(alpha=2.5)  # Higher focus on tails
model.fit(historical_features, historical_losses)

# Predict potential losses
portfolio_risk = model.predict(current_positions)

# Evaluate CVaR for regulatory compliance
cvar_95 = cvar_loss(actual_losses, predictions, alpha=0.95)
```

Impact:

- Improved capital allocation
- Regulatory compliance (Basel III)
- Better stress testing capabilities

6.3 Healthcare Cost Prediction

Problem Statement:

Rare catastrophic cases (ICU admissions, complex surgeries) drive healthcare costs but are poorly predicted by standard models, leading to:

- Inaccurate premium pricing
- Unsustainable risk pools
- Unexpected losses for insurers

TailRisk Solution:

Focus on identifying and accurately pricing high-cost cases:

```python
from tailrisk import HybridMetaLearner
from tailrisk.utils import print_tail_validation

model = HybridMetaLearner(
    base_estimators=medical_models,
    quantile=0.95,
    blend_lambda=0.25
)

model.fit(patient_features, medical_costs)
cost_predictions = model.predict(new_patients)

# Evaluate tail performance
print_tail_validation(
    actual_costs,
    cost_predictions,
    model_name="Healthcare Costs"
)
```

7 Model Comparison Example

7.1 Complete Workflow

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import (
    RandomForestRegressor,
    GradientBoostingRegressor
)
from tailrisk import LaRRegressor, HybridMetaLearner
from tailrisk.utils import compare_models, print_tail_validation

# Generate heavy-tailed data
np.random.seed(42)
n_samples = 5000
X = np.random.randn(n_samples, 10)

# Exponential distribution (heavy tail)
y = np.random.exponential(scale=1000, size=n_samples)

# Add catastrophic claims
n_catastrophic = int(0.05 * n_samples)
catastrophic_idx = np.random.choice(
    n_samples,
    n_catastrophic,
    replace=False
)
y[catastrophic_idx] = np.random.exponential(
    scale=20000,
    size=n_catastrophic
)

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Train baseline model
baseline = LinearRegression()
baseline.fit(X_train, y_train)
y_pred_baseline = baseline.predict(X_test)

# Train LaR model
lar_model = LaRRegressor(alpha=2.0)
lar_model.fit(X_train, y_train)
y_pred_lar = lar_model.predict(X_test)

# Train Hybrid Meta-Learner
base_models = [
    ('rf', RandomForestRegressor(
        n_estimators=100,
        max_depth=10,
        random_state=42
    )),
    ('gb', GradientBoostingRegressor(
        n_estimators=100,
        max_depth=5,
        random_state=42
```

```
    ))
]

hybrid_model = HybridMetaLearner(
    base_estimators=base_models,
    quantile=0.95,
    blend_lambda=0.25,
    lar_alpha=1.5,
    cv_folds=5
)

hybrid_model.fit(X_train, y_train)
y_pred_hybrid = hybrid_model.predict(X_test)

# Compare all models
results = compare_models(y_test, {
    'Baseline (LR)': y_pred_baseline,
    'LaR (alpha=2.0)': y_pred_lar,
    'Hybrid Meta': y_pred_hybrid
})

# Detailed validation for best model
print_tail_validation(
    y_test,
    y_pred_hybrid,
    model_name="Hybrid Meta-Learner"
)
```

7.2 Expected Output

Table 1: Model Comparison Results

Metric	Baseline	LaR	Hybrid
MSE (Overall)	2,808,591	2,815,649	2,979,450
MSE (Extreme)	260,918,453	259,101,161	252,549,889
CVaR (95%)	3,146	3,086	2,824
Detection @ 90%	3.1%	3.2%	6.1%
Detection @ 95%	0.8%	0.9%	2.5%
TCR @ 95%	0.165	0.166	0.306
TCR @ 99%	0.058	0.059	0.102

Key Observations:

1. Hybrid model shows modest increase in overall MSE (+6%)
2. **Extreme MSE improves by 3%** (critical for tail risk)
3. **CVaR improves by 10%** (better worst-case performance)
4. **TCR@99% nearly doubles** (+76%, from 0.058 to 0.102)
5. **Detection rates triple** (0.8% → 2.5% @ 95%)

Interpretation:

The Hybrid Meta-Learner trades a small increase in average error for dramatic improvements in tail risk metrics. This is the desired behavior for applications where extreme events drive losses.

8 Best Practices

8.1 Model Selection

Table 2: Model Selection Guide

Model	When to Use	Advantages
LaRRegressor	• Simple problems • Limited data • Need interpretability	Fast training, interpretable, good baseline
HybridMetaLearner	• Complex patterns • Sufficient data • Critical applications	Best tail performance, robust, production-ready
CVaRWeightedEnsemble	• Multiple models • Model combination • Ensemble methods	Automatic weighting, leverages diversity

8.2 Hyperparameter Tuning

8.2.1 LaRRegressor Alpha Parameter

- $\alpha \in [1.0, 1.5]$: Moderate tail focus
- $\alpha \in [1.5, 2.5]$: Strong tail focus (recommended)
- $\alpha > 2.5$: Very aggressive (may sacrifice average performance)

8.2.2 HybridMetaLearner Configuration

- **quantile**:
  - 0.90 for moderate tail focus
  - 0.95 for standard tail risk (recommended)
  - 0.99 for extreme tail events only
- **blend_lambda**:
  - 0.15–0.25: Prioritize overall accuracy
  - 0.25–0.30: Balanced (recommended)
  - 0.30–0.40: Prioritize tail detection
- **Base Estimators**:
  - Minimum 2, recommended 3–5
  - Include diverse types (tree, linear, boosting)
  - Avoid too many correlated models

8.3 Data Considerations

8.3.1 Minimum Sample Size

Table 3: Recommended Sample Sizes

Model	Minimum Samples
LaRRegressor	200–500
HybridMetaLearner	1,000+
CVaRWeightedEnsemble	500–1,000

8.3.2 Train/Test Split Strategy

For tail risk applications, use **stratified splitting** to ensure test set contains representative extreme values:

```
from sklearn.model_selection import StratifiedKFold

# Create binary indicator for tail events
threshold = np.percentile(y, 95)
is_tail = (y >= threshold).astype(int)

# Stratified split
splitter = StratifiedKFold(
    n_splits=5,
    shuffle=True,
    random_state=42
)

for train_idx, test_idx in splitter.split(X, is_tail):
    X_train, X_test = X[train_idx], X[test_idx]
    y_train, y_test = y[train_idx], y[test_idx]
    break  # Use first fold
```

8.4 Evaluation

Always evaluate using tail-specific metrics:

1. **Primary metrics**: CVaR, TCR
2. **Secondary metrics**: Detection Rate, LaR
3. **For context**: Overall MSE, MAE

Never rely on MSE/MAE alone for tail risk applications!

9 Scikit-learn Integration

TailRisk is fully compatible with scikit-learn ecosystem:

9.1 GridSearchCV

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer
from tailrisk import LaRRegressor, cvar_loss

# Custom scorer for CVaR
def cvar_scorer(y_true, y_pred):
    return -cvar_loss(y_true, y_pred, alpha=0.95)

cvar_score = make_scorer(cvar_scorer, greater_is_better=True)

# Grid search
param_grid = {
    'alpha': [1.0, 1.5, 2.0, 2.5, 3.0]
}

grid = GridSearchCV(
    LaRRegressor(),
    param_grid,
    cv=5,
    scoring=cvar_score,
    verbose=1
)

grid.fit(X_train, y_train)
best_model = grid.best_estimator_
```

9.2 Pipeline

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from tailrisk import LaRRegressor

# Create pipeline
pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('model', LaRRegressor(alpha=2.0))
])

# Train
pipe.fit(X_train, y_train)

# Predict
predictions = pipe.predict(X_test)
```

9.3 Cross-Validation

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(
    LaRRegressor(alpha=2.0),
```

```
    X, y,
    cv=5,
    scoring='neg_mean_squared_error'
)

print(f"CV MSE: {-scores.mean():.2f} +/- {scores.std():.2f}")
```

10 Advanced Topics

10.1 Custom Base Estimators

Create domain-specific base models:

```python
from sklearn.base import BaseEstimator, RegressorMixin

class DomainSpecificModel(BaseEstimator, RegressorMixin):
    """Custom model incorporating domain knowledge"""

    def __init__(self, domain_param=1.0):
        self.domain_param = domain_param

    def fit(self, X, y, sample_weight=None):
        # Implement custom fitting logic
        # Must support sample_weight for LaR
        return self

    def predict(self, X):
        # Implement prediction logic
        return predictions

# Use in HybridMetaLearner
base_models = [
    ('rf', RandomForestRegressor()),
    ('custom', DomainSpecificModel(domain_param=2.0))
]

model = HybridMetaLearner(base_estimators=base_models)
```

10.2 Ensemble Stacking

Combine TailRisk models with traditional stacking:

```python
from sklearn.ensemble import StackingRegressor
from tailrisk import LaRRegressor, HybridMetaLearner

# Create stacking ensemble
stacking = StackingRegressor(
    estimators=[
        ('lar', LaRRegressor(alpha=2.0)),
        ('hybrid', HybridMetaLearner(
            base_estimators=base_models
        ))
    ],
    final_estimator=Ridge(alpha=1.0)
)

stacking.fit(X_train, y_train)
```

10.3 Production Deployment

10.3.1 Model Serialization

```python
import joblib

# Save model
```

```
joblib.dump(hybrid_model, 'production_model.pkl')

# Load model
loaded_model = joblib.load('production_model.pkl')

# Use in production
predictions = loaded_model.predict(new_data)
```

10.3.2 Monitoring

Monitor model performance in production:

```
def monitor_tail_performance(
    y_true,
    y_pred,
    alert_threshold_tcr=0.15
):
    """Monitor production model performance"""
    from tailrisk import tail_coverage_ratio

    tcr = tail_coverage_ratio(
        y_true,
        y_pred,
        quantile=0.99
    )

    if tcr < alert_threshold_tcr:
        print(f"WARNING: TCR dropped to {tcr:.3f}")
        print("Model may need retraining!")
        # Trigger alert/notification

    return tcr

# Use monthly or quarterly
tcr_current = monitor_tail_performance(
    recent_actuals,
    recent_predictions
)
```

11 Troubleshooting

11.1 Common Issues

11.1.1 Issue: Negative Predictions

Problem: Model predicts negative values for inherently positive targets (costs, claims).
Solution:

```
# Post-process predictions
predictions = np.maximum(predictions, 0)

# Or use log-transformation
from sklearn.compose import TransformedTargetRegressor

model = TransformedTargetRegressor(
    regressor=LaRRegressor(alpha=2.0),
    func=np.log1p,
    inverse_func=np.expm1
)
```

11.1.2 Issue: Poor Tail Performance

Possible causes and solutions:

1. **Insufficient tail samples**
   - Solution: Collect more data, especially extreme cases
   - Consider synthetic oversampling of tail events
2. **Alpha too low**
   - Solution: Increase α to 2.5 or 3.0
3. **Wrong base models**
   - Solution: Include non-linear models (trees, boosting)
   - Ensure model diversity

11.1.3 Issue: Training Too Slow

Solutions:

```
# Reduce cv_folds
model = HybridMetaLearner(
    base_estimators=base_models,
    cv_folds=3  # Instead of 5
)

# Use fewer/simpler base models
base_models = [
    ('rf', RandomForestRegressor(n_estimators=50)),  # Reduce trees
    ('ridge', Ridge())  # Fast linear model
]

# Parallelize base model training
model = RandomForestRegressor(
    n_estimators=100,
```

```
    n_jobs=-1  # Use all CPU cores
)
```

11.2 Debugging

Enable detailed output:

```
import logging

logging.basicConfig(level=logging.DEBUG)

# Training will show detailed progress
model.fit(X_train, y_train)
```

12 Theoretical Background

12.1 Quantile Regression

Quantile regression estimates conditional quantiles of the response variable:

$$Q_\tau(Y|X = x) = \arg\min_q \mathbb{E}[\rho_\tau(Y - q)|X = x] \tag{13}$$

where the check function (pinball loss) is:

$$\rho_\tau(u) = u(\tau - \mathbb{I}(u < 0)) = \begin{cases} \tau \cdot u & \text{if } u \geq 0 \\ (\tau - 1) \cdot u & \text{if } u < 0 \end{cases} \tag{14}$$

For tail risk, we focus on high quantiles ($\tau = 0.95, 0.99$) to capture extreme outcomes.

12.2 Extreme Value Theory

Heavy-tailed distributions (common in insurance and finance) satisfy:

$$\lim_{x \to \infty} \frac{\mathbb{P}(X > x + y|X > x)}{\mathbb{P}(X > y)} = 1 \tag{15}$$

This memoryless property makes standard regression challenging. TailRisk's weighted approaches account for this heavy-tail behavior.

12.3 CVaR as Coherent Risk Measure

CVaR satisfies the four axioms of coherent risk measures:

1. **Monotonicity**: If $X \leq Y$, then $\text{CVaR}(X) \leq \text{CVaR}(Y)$
2. **Translation invariance**: $\text{CVaR}(X + c) = \text{CVaR}(X) + c$
3. **Homogeneity**: $\text{CVaR}(\lambda X) = \lambda \cdot \text{CVaR}(X)$ for $\lambda \geq 0$
4. **Subadditivity**: $\text{CVaR}(X + Y) \leq \text{CVaR}(X) + \text{CVaR}(Y)$

This makes CVaR superior to VaR for risk management applications.

13 References

13.1 Package Documentation

- **GitHub Repository**: https://github.com/VishalLakshmiNarayanan/TailriskLib
- **PyPI Page**: https://pypi.org/project/tailrisk/
- **API Reference**: https://github.com/VishalLakshmiNarayanan/TailriskLib/blob/main/docs/API_REFERENCE.md
- **Tutorial**: https://github.com/VishalLakshmiNarayanan/TailriskLib/blob/main/docs/TUTORIAL.md

13.2 Related Software

- **Scikit-learn**: https://scikit-learn.org/
- **StatsModels**: https://www.statsmodels.org/ (Quantile Regression)
- **PyRisk**: https://github.com/quantopian/pyfolio (Financial Risk Analysis)

14 Citation

If you use TailRisk in academic work, please cite:

```
@software{tailrisk2025,
  author = {Vishal Lakshmi Narayanan},
  title = {TailRisk: Risk-Aware Machine Learning
           for Tail Risk Modeling},
  year = {2025},
  version = {0.1.2},
  url = {https://github.com/VishalLakshmiNarayanan/TailriskLib},
  doi = {10.5281/zenodo.XXXXXXX}
}
```

15 License

TailRisk is released under the MIT License:

16 Contact & Support

16.1 Author

Vishal Lakshmi Narayanan

- Email: lvishal1607@gmail.com
- GitHub: @VishalLakshmiNarayanan
- Repository: `https://github.com/VishalLakshmiNarayanan/TailriskLib`

16.2 Bug Reports

Report bugs at: `https://github.com/VishalLakshmiNarayanan/TailriskLib/issues`

Please include:

- Python version
- TailRisk version
- Minimal reproducible example
- Expected vs. actual behavior
- Full error traceback

16.3 Feature Requests

Submit feature requests as GitHub issues with:

- Use case description
- Proposed API (if applicable)
- Relevance to tail risk modeling

16.4 Contributing

Contributions are welcome! See `CONTRIBUTING.md` in the repository for:

- Development setup
- Code style guidelines
- Pull request process
- Testing requirements

TailRisk v0.1.2 | 2025