Custom Metrics

The tree_machine library now supports custom metrics alongside the pre-defined ones. This allows you to use your own evaluation functions while maintaining compatibility with existing predefined metrics.

Overview

You can now pass either: - **String names** for predefined metrics (existing functionality) - **Custom functions** for your own metrics

Usage Examples

Classification with Custom Metrics

```
import numpy as np
from sklearn.metrics import accuracy_score, log_loss
from sklearn.model_selection import StratifiedKFold
from tree_machine import ClassifierCV, default_classifier

# Using predefined metrics (existing functionality)
clf_predefined = ClassifierCV(
    metric="f1_macro",  # String name
    cv=StratifiedKFold(n_splits=5),
    n_trials=10,
    timeout=60,
    config=default_classifier
)

# Using custom metrics
def custom_accuracy(y_true, y_pred):
    """Custom accuracy metric."""
    return accuracy_score(y_true, y_pred)

def custom_log_loss(y_true, y_pred):
    """Custom log loss metric (for probability predictions)."""
    return -log_loss(y_true, y_pred)  # Negative because we want to maximize

# Custom metric with additional parameters
def weighted_f1(y_true, y_pred, beta=1.0):
    """Custom weighted F1 score."""
    from sklearn.metrics import fbeta_score
    return fbeta_score(y_true, y_pred, beta=beta)

# Using custom metrics
clf_custom = ClassifierCV(
    metric=custom_accuracy,  # Custom function
    cv=StratifiedKFold(n_splits=5),
    n_trials=10,
    timeout=60,
    config=default_classifier
)

# Using sklearn metrics directly
clf_sklearn = ClassifierCV(
    metric=accuracy_score,  # Direct sklearn function
    cv=StratifiedKFold(n_splits=5),
    n_trials=10,
    timeout=60,
    config=default_classifier
)
```

Regression with Custom Metrics

```
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import KFold
from tree_machine import RegressionCV, default_regression

# Using predefined metrics (existing functionality)
reg_predefined = RegressionCV(
    metric="mae",  # String name
    cv=KFold(n_splits=5),
    n_trials=10,
    timeout=60,
    config=default_regression
)

# Using custom metrics
def custom_r2(y_true, y_pred):
    """Custom R² score."""
    return r2_score(y_true, y_pred)

def custom_mape(y_true, y_pred):
    """Custom Mean Absolute Percentage Error."""
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

def custom_huber_loss(y_true, y_pred, delta=1.0):
    """Custom Huber loss."""
    residual = np.abs(y_true - y_pred)
    return np.where(
        residual <= delta,
        0.5 * residual**2,
        delta * residual - 0.5 * delta**2
    ).mean()

# Using custom metrics
reg_custom = RegressionCV(
    metric=custom_r2,  # Custom function
    cv=KFold(n_splits=5),
    n_trials=10,
    timeout=60,
    config=default_regression
)

# Using sklearn metrics directly
reg_sklearn = RegressionCV(
    metric=r2_score,  # Direct sklearn function
    cv=KFold(n_splits=5),
    n_trials=10,
    timeout=60,
    config=default_regression
)
```

Custom Metric Function Requirements

Your custom metric function must:

1. **Accept exactly two parameters**: `y_true` and `y_pred`
2. **Return a single float value**: The metric score
3. **Handle numpy arrays**: Both parameters will be numpy arrays
4. **Be deterministic**: Same inputs should always produce the same output

Example Custom Metric Function

```
def my_custom_metric(y_true, y_pred):
    """
    Custom metric function.

    Args:
        y_true: Ground truth values (numpy array)
        y_pred: Predicted values (numpy array)

    Returns:
        float: Metric score
    """
    # Your custom calculation here
    error = np.abs(y_true - y_pred)
    return np.mean(error)  # Example: mean absolute error
```

Error Handling

If you provide an invalid metric, you'll get a clear error message:

```
# This will raise a ValueError with helpful message
clf = ClassifierCV(
    metric="invalid_metric",  # Not in predefined metrics
    cv=StratifiedKFold(n_splits=5),
    n_trials=10,
    timeout=60,
    config=default_classifier
)
# Error: Unknown classification metric 'invalid_metric'.
# Available predefined metrics: f1, f1_macro, f1_micro, ...
# You can also pass a custom metric function.
```

Backward Compatibility

All existing code using string-based metrics will continue to work without any changes:

```
# This still works exactly as before
clf = ClassifierCV(
    metric="f1_macro",  # Predefined metric
    cv=StratifiedKFold(n_splits=5),
    n_trials=10,
    timeout=60,
    config=default_classifier
)
```

Available Predefined Metrics

Classification Metrics

- `f1`, `f1_macro`, `f1_micro`, `f1_samples`, `f1_weighted`
- `precision`, `precision_macro`, `precision_micro`, `precision_samples`, `precision_weighted`
- `recall`, `recall_macro`, `recall_micro`, `recall_samples`, `recall_weighted`

Regression Metrics

- `mae` (Mean Absolute Error)
- `mape` (Mean Absolute Percentage Error)
- `median` (Median Absolute Error)
- `mse` (Mean Squared Error)
- `quantile` (Mean Pinball Loss)

Tips for Custom Metrics

1. **Consider the optimization direction**: The library automatically handles `greater_is_better` for classification (True) and regression (False)
2. **Test your metric**: Make sure it works with your data types and edge cases

3. **Document your metric**: Add docstrings explaining what your metric measures
4. **Use vectorized operations**: Leverage numpy for efficient calculations
5. **Handle edge cases**: Consider what happens with NaN values, empty arrays, etc.

Advanced Example: Domain-Specific Metric

```
def business_metric(y_true, y_pred):
    """
    Custom business metric that weighs different types of errors differently.
    """
    error = y_true - y_pred

    # Over-prediction is more costly than under-prediction
    over_prediction_penalty = np.where(error < 0, np.abs(error) * 2, 0)
    under_prediction_penalty = np.where(error > 0, error * 1, 0)

    total_cost = np.sum(over_prediction_penalty + under_prediction_penalty)
    return -total_cost  # Negative because we want to minimize cost

# Use in regression
reg_business = RegressionCV(
    metric=business_metric,
    cv=KFold(n_splits=5),
    n_trials=10,
    timeout=60,
    config=default_regression
)
```