Deep Regression Trees (Experimental)

Tree Machine includes an **experimental** regressor called `DFRegression`.

It builds a neural-network model that behaves like a small forest of decision trees. The goal is to give you a tree-like model that can be trained with the same tools you use for neural networks.

This feature lives under `tree_machine.deep_trees`.

When to use it

Use `DFRegression` when:

- you want a regression model with a tree-style structure;
- you want to train with TensorFlow / Keras training loops (`Model.fit`);
- you want to experiment with regularization and dropout options that are common in neural-network training.

If you want automated model selection and hyperparameter tuning, prefer `RegressionCV`.

Requirements

`DFRegression` needs TensorFlow.

If you installed Tree Machine without TensorFlow, install it separately:

```
pip install tensorflow
```

Basic usage

`DFRegression` follows the scikit-learn style API: `fit`, `predict`, and `score`.

```
import numpy as np
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split

from tree_machine.deep_trees.regression import DFRegression

X, y = make_regression(n_samples=2000, n_features=20, noise=0.5, random_state=0)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=0)

model = DFRegression(
    metric="mse",
    n_estimators=32,
    internal_size=64,
    max_depth=4,
    feature_fraction=0.8,
)

model.fit(
    X_train,
    y_train,
    epochs=20,
    batch_size=256,
    verbose=0,
)

pred = model.predict(X_test)
print(pred[:5])

# "score" returns the negative loss value (higher is better)
print(model.score(X_test, y_test))
```

Parameters you will likely tune

`metric`

Controls the built-in loss/metric pairing used when you do not provide a custom loss.

Supported values:

- `"mae"` (mean absolute error)
- `"mse"` (mean squared error)
- `"mape"` (mean absolute percentage error)

Forest and tree size

- `n_estimators`: number of trees in the forest.
- `max_depth`: depth of each tree. Larger values increase model capacity and memory use.
- `internal_size`: the size of the internal representation used inside each tree.

Feature sampling

- `feature_fraction`: fraction of input features used per tree. Lower values can reduce overfitting.

Regularization and dropout

These settings can help control overfitting.

- `decision_l1`, `decision_l2`: regularization for the routing (split) part of the tree.
- `leaf_l1`, `leaf_l2`: regularization for the leaf values.
- `feature_dropout`: dropout applied to the inputs during training.
- `routing_dropout`: dropout applied to the routing probabilities during training.

Custom loss and metrics

You can provide your own Keras loss and metrics.

Rules:

- If you pass both `loss` and `metrics`, the built-in `metric` key is only used for validation.
- If you pass `metrics` but not `loss`, the built-in loss derived from `metric` is used.

Example:

```
import tensorflow as tf
from tree_machine.deep_trees.regression import DFRegression

model = DFRegression(
    metric="mse",
    n_estimators=16,
    internal_size=32,
    max_depth=3,
    feature_fraction=1.0,
    loss=tf.keras.losses.Huber(),
    metrics=[tf.keras.metrics.RootMeanSquaredError()],
    compile_kwargs={"optimizer": "adam"},
)
```

Input formats

`X` can be:

- a NumPy array (`shape == (n_samples, n_features)`), or
- a pandas DataFrame.

If you pass a DataFrame, the column names seen in `fit` are stored and used to re-order columns in later calls.

`y` must be a 1D numeric array-like.

Notes and limitations

- This is an experimental API and may change between releases.
- Each call to `fit` builds and compiles a new Keras model.
- `score` returns the negative loss value on the provided data.
- Multi-output regression is not supported.